Bevezetés a számítástechnikába (jegyzet)

Bérci Norbert, Uhlár László, Tuza Zoltán

2011/2012 őszi félév
v0.1

Tartalomjegyzék

1. fejezet

Számok és karakterek ábrázolása

Bérci Norbert

1.1. Számrendszerek

A számrendszerek [numeral system - not numeric system!] a szám (mint matematikai fogalom) írott formában történő megjelenítésére alkalmas módszerek. Ebben a részben a helyiértéken (pozíción) alapuló számrendszereket tárgyaljuk. Léteznek nem pozíción alapuló számrendszerek is, ilyenek például a sorrendiségen alapuló római számok, de ezekkel a továbbiakban nem foglalkozunk.

1.1.1. A számrendszer alapja és a számjegyek

A helyiértéken alapuló számrendszerek két legfontosabb paramétere *a számrendszer alapja* [base] és *az egyes pozíciókba írható számjegyek* [digit]. Ezek nem függetlenek: a számrendszer alapja meghatározza az egyes pozíciókba írható számjegyek maximumát is: ha a számrendszer A alapú, akkor a legkisebb számjegy értéke 0 a legnagyobb számjegy értéke $A-1$.

1.1.1. példa. A tízes számrendszerben a $0, 1, 2, 3, 4, 5, 6, 7, 8, 9$ számjegyeket használhatjuk fel, a nyolcas számrendszerben a $0, 1, 2, 3, 4, 5, 6, 7$ számjegyek közül választhatunk, míg a kettesben a $0, 1$ a két lehetséges számjegy.

Tíznél nagyobb alapú számrendszerek esetében a számjegyek halmazát 9 után az ABC betűivel egészítjük ki. A kis és nagybetűk között általában nem teszünk különbséget, bár egyes nagy alapú számrendszereknél erre mégis szükség lehet.

1.1.2. példa. A tizenhatos számrendszerben használható "számjegyek": 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f (vagy 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F).

Ha az a szövegkörnyezetből nem egyértelmű, a számrendszer alapját szögletes zárójelben a jobb alsó indexbe téve jelölhetjük. Például: $5221_{[10]}$, $726_{[8]}$ vagy $80_{[16]}$.

1.1.2. Alaki- és helyiérték

Egy adott számrendszerben leírt szám esetében egy *számjegy értéke* egyenlő a számjegy *alaki értéké*nek és *helyiértéké*nek szorzatával. A számjegy alaki értéke a számjegyhez tartozó érték, a helyiérték pedig a számrendszer alapjának a pozíció szerinti hatványa. A $0, 1, \ldots, 9$ esetében az alaki érték egyértelmű, a betűkkel kiegészített esetben pedig: a=10, b=11, c=12, d=13 stb.

1.1.3. példa. A tízes számrendszerben felírt 32 szám esetében a 3 helyiértéke $10^1 = 10$, mivel az jobbról a második pozíción szerepel (és a helyiértékeket a nulladik hatványtól indítjuk), így értéke: $3 \cdot 10^1 = 3 \cdot 10 = 30$.

1.1.4. példa. A tízes számrendszerben felírt 32 szám esetében a 2 helyiértéke $10^0 = 1$, mivel az jobbról az első pozíción szerepel (és a helyiértékeket a nulladik hatványtól indítjuk), így értéke: $2 \cdot 10^0 = 2 \cdot 1 = 2$.

A jól ismert tízes alapú *decimális* számrendszeren kívül az informatikában a leggyakrabban használtak a következők: a kettes alapú *bináris*, a nyolcas alapú *oktális* és a tizenhatos alapú *hexadecimális*. Az előzőekben említett, indexben történő számrendszer megadás mellett oktális számrendszer jelölésére használatos a kezdő 0 szerepeltetése, hexadecimális számok esetén a 0x, 0X prefixek vagy a h postfix. Az informatikában ezeket a jelöléseket használjuk a leginkább. Például: 065 (oktális), 0x243 (hexadecimális), 0X331 (hexadecimális), 22h (hexadecimális). Ha sem a szám előtt, sem utána, sem az indexében nincs jelölve, akkor decimális számrendszerben értelmezzük a számot.

1.1.3. Egész számok leírása

Egész számokat általános esetben az $a_n a_{n-1} \dots a_1 a_0$ alakban írhatunk fel, és az így felírt szám értéke (A alapú számrendszert feltételezve):

$$(a_n \cdot A^n) + (a_{n-1} \cdot A^{n-1}) + \dots + (a_1 \cdot A^1) + (a_0 \cdot A^0)$$

ami nem más, mint a leírt számjegyek az előzőekben megismert módon kiszámolt értékeinek (alaki érték szorozva a helyiértékkel) összege.

1.1.5. példa. Triviális példa: $405_{[10]} = 4 \cdot 10^2 + 0 \cdot 10^1 + 5 \cdot 10^0 = 400 + 5$

1.1.6. példa. $405_{[8]} = 4 \cdot 8^2 + 0 \cdot 8^1 + 5 \cdot 8^0 = 256 + 5 = 261_{[10]}$

1.1.7. példa. $1001101_{[2]} = 1 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 64 + 8 + 4 + 1 = 77_{[10]}$

1.1.8. példa. $0\text{xA3} = 10 \cdot 16^1 + 3 \cdot 16^0 = 10 \cdot 16 + 3 \cdot 1 = 163_{[10]}$

1.1.4. Nem egész számok leírása

Az egész számoknál megismert felírási módszert kiterjeszthetjük úgy, hogy a helyiértékek megadásánál nem állunk meg a nulladik hatványnál, hanem folytatjuk azt a negatív hatványokra is, így lehetőségünk adódik nem egész számok leírására. Általános esetben tehát ennek alakja: $a_n a_{n-1} \dots a_1 a_0 a_{-1} \dots a_{-k}$, és az így felírt szám értéke (A alapú számrendszert feltételezve):

$$a_n \cdot A^n + a_{n-1} \cdot A^{n-1} + \dots + a_1 \cdot A^1 + a_0 \cdot A^0 + a_{-1} \cdot A^{-1} + \dots + a_{-k} \cdot A^{-k}$$

Annak érdekében, hogy a mindkét végén (egész illetve törtrész) tetszőlegesen bővíthető felírás egyértelmű legyen, ennek a két résznek a határát jelöljük tizedesponttal (illetve a magyar helyesírás szerint tizedesvesszővel).[1] Mi a továbbiakban a tizedespontos jelölést fogjuk alkalmazni.

1.1.9. példa. Triviális példa: $405.23_{[10]} = 4 \cdot 10^2 + 0 \cdot 10^1 + 5 \cdot 10^0 + 2 \cdot 10^{-1} + 3 \cdot 10^{-2} = 4 \cdot 100 + 5 \cdot 1 + 2 \cdot \frac{1}{10} + 3 \cdot \frac{1}{100}$

1.1.10. példa. $405.23_{[8]} = 4 \cdot 8^2 + 0 \cdot 8^1 + 5 \cdot 8^0 + 2 \cdot 8^{-1} + 3 \cdot 8^{-2} = 4 \cdot 64 + 5 \cdot 1 + 2 \cdot \frac{1}{8} + 3 \cdot \frac{1}{8^2} = 256 + 5 + \frac{2}{8} + \frac{3}{64} = 261.296875_{[10]}$

1.1.11. példa. $1001101.01_{[2]} = 1 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2} = 64 + 8 + 4 + 1 + \frac{1}{4} = 77.25_{[10]}$

1.1.5. Átváltás számrendszerek között

Az adott számrendszerből tízes számrendszerbe váltást az előzőekben hallgatólagosan már bemutattuk. A fordított átváltásra nem térünk ki (a módszer könnyen kitalálható).

Az átváltás nagymértékben egyszerűsödik, ha a binárisból oktális vagy hexadecimális számrendszerbe kell átváltani: egyszerűen hármasával (oktális esetben) vagy négyesével (hexadecimális esetben) kell a bináris számjegyeket csoportosítani, és az így képzett csoportokat átváltani. Az átváltás fordított irányban is hasonlóan egyszerű: az egyes oktális vagy hexadecimális számjegyeket kell átváltani és az így kapott hármas illetve négyes bináris csoportokat egymás után írni. Oktálisból hexadecimálisba vagy decimálisból hexadecimálisba illetve fordítva a bináris számrendszert közbeiktatva egyszerűen átválthatunk.

[1] Ha nagyon pontosak akarunk lenni, akkor tizedespontról csak a tízes számrendszer használata esetén beszélhetnénk, bináris esetben inkább bináris pontról van szó (és hasonlóan oktális, hexadecimális stb. esetben).

1.1.12. példa. $110111101_{[2]} = 110\ 111\ 101_{[2]} = 675_{[8]}$

1.1.13. példa. $10111101_{[2]} = 1011\ 1101_{[2]} =$0xbd

1.1.14. példa. $\mathrm{c6d}_{[16]} = 1100\ 0110\ 1101_{[2]} = 110\ 001\ 101\ 101_{[2]} = 06155$

1.1.6. Feladatok

1.1.1. feladat. $1010111001_{[2]} = ?_{[8]} = ?_{[16]}$

1.1.2. feladat. $54_{[8]} = ?_{[16]}$

1.1.3. feladat. $962_{[10]} = ?_{[8]} = ?_{[16]}$

1.1.4. feladat. $\mathrm{9a2d}_{[16]} = ?_{[2]} = ?_{[8]} = ?_{[16]}$

1.1.5. feladat. Adjunk algoritmust (módszert) decimálisból a) oktális-, b) hexadecimális számrendszerbe történő közvetlen (tehát nem a bináris számrendszer közbeiktatásával történő) átváltásra!

1.1.6. feladat. Minden racionális szám (tört) leírható bármilyen alapú számrendszerben véges számjegy felhasználásával?

1.1.7. feladat. A http://www.digitconvert.com/ oldalon kipróbálhatók, ellenőrizhetők az átváltások.

1.1.7. Számrendszerek pontossága

Fontos kiemelni, hogy nem egész számok felírása esetén nem biztos, hogy a szám pontosan leírható véges számjeggyel! Máshogyan megfogalmazva, a nem egész számok ábrázolásának pontossága függ a számrendszer alapjától: például az $\frac{1}{3}$ tízes számrendszerben nem írható fel véges számjeggyel, ugyanakkor hármas számrendszerben pontosan felírható: $\frac{1}{3} = 0.1_{[3]} = 0.33333\ldots_{[10]}$

1.1.8. feladat. Adjunk meg néhány példát arra, amikor az egyik számrendszerben véges számjeggyel felírható szám a másik számrendszerben nem írható fel véges számjeggyel!

1.1.9. feladat. Adjunk meg néhány példát olyan számra, ami egyetlen számrendszerben sem írható fel véges számjeggyel!

1.1.10. feladat. Kiválasztható olyan alapú számrendszer, amiben minden racionális szám pontosan ábrázolható? Indokoljuk meg!

1.2. Mértékegységek

Az informatikában használatos legkisebb egység a bit (sok esetben b-vel rövidítik, de a legfrissebb szabvány[2] a rövidítés nélküli formát ajánlja). Értéke 0 vagy 1 lehet. Használhatjuk tárolókapacitás vagy információ jelölésére. Az utóbbi egy felsőbb éves tárgy, az *Információ és kódelmélet* témája, mi itt csak a tárolási vonatkozásával foglalkozunk.

A bájt (byte) az informatika másik legfontosabb egysége, jele: B. Mi az általánosan elfogadott, a gyakorlatban majdnem kizárólagosan használt 1 B = 8 bit átváltást használjuk, bár egyes (egzotikus) architektúrák esetében ennél több vagy kevesebb bit is alkothat egy bájtot.

Az SI mértékegységrendszerben használatos k (kilo), M (mega), G (giga), T (tera), P (peta) stb. prefixek mellett a bit és a bájt esetében használatosak a Ki (kibi), Mi (mebi), Gi (gibi), Ti (tebi), Pi (pebi) stb. *bináris prefixek* is (lásd az 1.1. ábrán). Fontos kiemelni, hogy az egyre nagyobb prefixek esetében egyre nagyobb a különbség az SI és az informatikai/bináris prefixek között. Például a G (1000^3) és Gi (1024^3) között a különbség kb. 7%, a T (1000^4) és Ti (1024^4) között már kb. 10%.[3]

[2] ISO/IEC 80000, Part 13 - Information science and technology

[3] Különösen fontos ez a háttértárak esetében, ahol a gyártók inkább az SI prefixeket használják, mert így egy 1000000000000 B méretű lemezegység esetében 1 TB-ot tüntethetnek fel, míg ugyanez a bináris prefixekkel csupán 0.9 TiB

A kapcsolat a prefixek és a számrendszerek között ott fedezhető fel, hogy a használt prefixek mindig a számrendszer alapja valamely hatványának hatványai. Az SI esetben ez a tíz harmadik hatványa (illetve ennek további hatványai), de ugyanez igaz a bináris prefixekre is, amikor is ez a kettő tizedik hatványa (illetve ennek további hatványai).

prefix	szorzó	prefix	szorzó
k (kilo)	1000	Ki (kibi)	1024
M (mega)	1000^2	Mi (mebi)	1024^2
G (giga)	1000^3	Gi (gibi)	1024^3
T (tera)	1000^4	Ti (tebi)	1024^4
P (peta)	1000^5	Pi (pebi)	1024^5

1.1. ábra. SI és bináris prefixek

1.3. Gépi számábrázolás

A gépi számábrázolás a számok (számító)gépek memóriájában vagy háttértárán történő tárolását jelenti.

1.3.1. Előjel nélküli egész számok ábrázolása

Az előjel nélküli (nem negatív) egész számok [unsigned integer] ábrázolása megegyezik a bináris számrendszernél megismert leírással: $86_{[10]} = 01010110_{[2]}$, azaz egy nem negatív egész számot a kettes számrendszerbe átváltott formájában tárolunk. A tömörebb írásmód miatt ugyanakkor ezt legtöbbször nem bináris, hanem hexadecimális formában írjuk le. (Ne feledjük, hogy a bináris - hexadecimális átváltás nem más, mint négy bitesével csoportosítás, ahogy azt az előzőekben láthattuk.)

Mivel a gyakorlatban általában csak egész byte méretű ábrázolásokat használunk, az előjel nélküli egészek is legtöbbször 1, 2, 4, 8, 16 byte (8, 16, 32, 64, 128 bit) hosszúak lehetnek. Így is hívjuk ezeket: 8 bites előjel nélküli egész, 16 bites előjel nélküli egész stb.

1.3.1. példa. A $46_{[10]}$ számot a memóriában a következőképpen tároljuk 1 bájton: 00101110.

1.3.1. feladat. Az összeadás művelet hogyan végezhető el az előjel nélküli egész számok bináris tárolása esetén? Adjunk erre módszert (algoritmust)!

1.3.2. Előjeles egész számok ábrázolása

Az előjeles egész számok [signed integer] ábrázolására az első gondolatunk az lehet, hogy az előjel nélküli egészek ábrázolásához egy előjelet jelentő bit hozzáadásával (ami például 0, ha pozitív az előjel és 1, ha negatív az előjel) egyszerűen meg lehet oldani a problémát. Sajnos azonban ez a megoldás sok szempontból nem megfelelő: a legkézenfekvőbb probléma, hogy ezzel a módszerrel lehetséges a $+0$ és a -0 ábrázolása is, ami zavarhoz vezet (például a „nulla-e” vizsgálatot így két különböző értékre kell megtenni), továbbá az ilyen módon felírt számokkal végzett műveletek bonyolultabbak, mint amennyire az feltétlenül szükséges lenne.

1.3.2. feladat. Az 1.3.1. feladatban kitalált összeadás művelet elvégezhető-e *módosítás nélkül* az előjelbites számábrázolási módszer használatával?

Sokkal jobb eredményre vezet a *kettes komplemens* ábrázolás: ahelyett, hogy egy előjelbittel jelölnénk az előjelet, a negatív számokat úgy ábrázoljuk, hogy hozzáadjuk őket egy nagy pozitív számhoz, és az eredményt ábrázoljuk, mint egy előjel nélküli egész számot. Ahhoz, hogy az eredmény biztosan pozitív legyen (hogy előjel nélküli egészként felírhassuk) de ugyanakkor ne is legyen túl nagy (hogy minél kevesebb biten felírható legyen az eredmény), a következő módszert alkalmazzuk: ha N biten akarjuk tárolni a kettes komplemens számot, akkor a negatív számhoz hozzáadandó nagy pozitív szám legyen a következő: 2^N.

1.3.2. példa. A -2 kettes komplemens ábrázolása 8 biten: $2^8 + (-2) = 256 - 2 = 254$, azaz: 11111110.

1.3.3. példa. A -17 kettes komplemens ábrázolása 8 biten: $2^8 + (-17) = 256 - 17 = 239$, azaz: 11101111.

A kettes komplemens ábrázolást az előző módszeren kívül megkaphatjuk úgy is, hogy a negatív számhoz egyet hozzáadunk, az eredmény abszolút értékét binárisan ábrázoljuk a megadott biten, és a számjegyeket invertáljuk.

1.3.4. példa. A -2 kettes komplemens ábrázolása 8 biten: $-2 + 1 = -1$ ennek abszolút értéke: $1 = 00000001$, invertálva: 11111110.

1.3.5. példa. A -17 kettes komplemens ábrázolása 8 biten: $-17 + 1 = -16$ ennek abszolút értéke: $16 = 00010000$, invertálva: 11101111.

Fontos tudnivalók:

- Kettes komplemens ábrázolásban is lehetséges nem negatív számok ábrázolása, aminek módja megegyezik az előjel nélküli egészek tárolási módjával. (Azaz ebben az esetben nem kell az előzőekben ismertetett műveleteket elvégezni.)
- A kettes komplemens ábrázolásban már csak egyetlen ábrázolási módja van a nullának.

1.3.3. feladat. Adjuk meg a 0 kettes komplemens ábrázolását 8, 16, 32, 64 biten!

1.3.4. feladat. Adjuk meg a -1 kettes komplemens ábrázolását 8, 16, 32, 64 biten!

1.3.5. feladat. Adjuk meg az 1 kettes komplemens ábrázolását 8 biten!

1.3.6. feladat. Az 1.3.1. feladatban kitalált összeadás művelet elvégezhető-e *módosítás nélkül* a kettes komplemens számábrázolási módszer használatával? Adjuk össze az előző két feladatban kiszámolt, 8 bites -1 és 1 értéket, és ellenőrizzük, hogy nullát kaptunk-e!

1.3.3. Egész számok ábrázolási határai

Az N biten történő, előjel nélküli egész számábrázolás esetén a tárolható legkisebb érték: 0, a tárolható legnagyobb érték: $2^N - 1$.

1.3.6. példa. Ha 8 bites előjel nélküli egész ábrázolást használunk, akkor a legkisebb ábrázolható szám a 00000000 (értéke 0), a legnagyobb ábrázolható szám az 11111111 (értéke $2^8 - 1 = 255$).

1.3.7. feladat. Mennyi a legnagyobb tárolható érték 8, 16, 32, 64 bites előjel nélküli egész esetében?

1.3.8. feladat. Összesen hány különböző érték tárolható 8, 16, 32, 64 biten, előjel nélküli egész számábrázolás esetében?

Ha kettes komplemens módon ábrázolunk egy egész számot és ehhez N bit áll rendelkezésre, akkor a tárolható legkisebb érték: -2^{N-1}, a tárolható legnagyobb érték: $2^{N-1} - 1$.

1.3.7. példa. Ha 8 bites kettes komplemens ábrázolást használunk, akkor a legkisebb ábrázolható szám az 10000000 (értéke -128), a legnagyobb ábrázolható szám a 01111111 (értéke 127).

1.3.9. feladat. Kettes komplemens ábrázolás esetén miért nem ugyanannyi szám tárolható a pozitív és a negatív tartományban? (Azaz miért nem -127 és 127 vagy -128 és 128 a két határ?)

1.3.10. feladat. Mennyi az értéke a kettes komplemens ábrázolással, 8 biten tárolt 11111111 illetve a 00000000 számoknak?

1.3.11. feladat. Eldönthető egyszerűen (ránézésre) egy kettes komplemens módon ábrázolt számról, hogy az negatív vagy pozitív?

1.3.12. feladat. Összesen hány különböző érték tárolható 8, 16, 32, 64 biten, kettes komplemens számábrázolás esetében?

1.3.13. feladat. Mi a kapcsolat az 1.3.8. feladat és az 1.3.12. feladatban kapott eredmények között?

1.3.4. Túlcsordulás

Az egész számok véges biten történő ábrázolása miatt mindig van legkisebb és legnagyobb ábrázolható szám. Amikor műveletet végzünk, elképzelhető, hogy a művelet eredménye már nem ábrázolható az operandusokkal megegyező méretben. Ezt a jelenséget túlcsordulásnak [overflow] nevezzük.

1.3.8. példa. Ha 8 bites előjel nélküli egészekkel dolgozunk, a 156+172=328 összeget már nem tudjuk 8 biten tárolni (mert a legnagyobb tárolható érték a 255).

1.3.9. példa. Ha 8 bites előjeles egészekkel dolgozunk, a -84+(-79)=-163 összeget már nem tudjuk 8 biten tárolni (mert a legkisebb tárolható érték a -127).

Túlcsordulás esetén – megvalósítástól függően – lehetséges

- *levágás*: a túlcsordult eredmény még ábrázolható részét tároljuk (a számláló kvázi körbefordul, mint például egy gázóránál, aminél fix számú helyiértéken történik a mérés),
- *szaturáció*: a túlcsordult eredmény helyett a legnagyobb illetve legkisebb ábrázolható értéket tároljuk.

1.3.10. példa. Levágás: ha 8 bites előjel nélküli egészekkel dolgozunk, a $156_{[10]} = 10011100_{[2]}$ és a $172_{[10]} = 10101100_{[2]}$ valódi összege ($328_{[10]} = 101001000_{[2]}$) helyett annak a 8 utolsó bitjét tároljuk: 01001000.

1.3.11. példa. Szaturáció: ha 8 bites előjel nélküli egészekkel dolgozunk, a $156_{[10]} = 10011100_{[2]}$ és a $172_{[10]} = 10101100_{[2]}$ valódi összege ($328_{[10]} = 101001000_{[2]}$) helyett az ábrázolható legnagyobb számot tároljuk: 11111111.

1.3.14. feladat. Mi az Y2K probléma? Mi a kapcsolat a túlcsordulás és az Y2K probléma között?

1.3.5. Lebegőpontos számábrázolás

Nem egész számok gépi ábrázolására a lebegőpontos [floating point] ábrázolást használjuk: a számot először átalakítjuk normalizált alakba, és az így kapott alak különböző részeit különkülön tároljuk.

Egy szám normalizált alakján olyan szorzatra bontását értjük, ahol a második tag a számrendszer alapjának valamely hatványa (amit a szám nagyságrendjének is nevezünk), az első tag értéke pedig annyi, hogy a második taggal megszorozva az eredeti számot kapjuk, továbbá az első tag legalább 1, de a számrendszer alapjánál kisebb értékű.

1.3.12. példa. Tízes számrendszerben a 382 normalizált alakja: $3.82 \cdot 10^2$, a 3.875 normalizált alakja $3.875 \cdot 10^0$, a 0.00000651 normalizált alakja $6.51 \cdot 10^{-6}$.

Az előzőekben definiált első tagot a szám *mantisszájának* [mantissa, significand], a hatványkitevőt (a nagyságrendet) a szám *karakterisztikájának* vagy *exponens*ének [exponent] nevezzük. Negatív számok tárolásához szükség van még az előjelre (ami 1 bit méretű, értéke 0, ha pozitív a szám és 1, ha negatív a szám). Egy szám ábrázolásához tehát ezeket az értékeket kell eltárolni.

Kettes számrendszerben ábrázolva a számot, a normalizált alak tovább egyszerűsödik, hiszen a mantissza tizedespontja előtt mindig 1 áll („az első tag legalább 1, de a számrendszer alapjánál kisebb értékű” feltétel miatt), amit így nem kell eltárolni, és ezt a megtakarított bitet a mantissza pontosabb tárolására lehet fordítani. Fontos, hogy a lebegőpontos szám értelmezésekor ezt az el nem tárolt számjegyet is figyelembe vegyük!

A lebegőpontos elnevezés abból adódik, hogy az ábrázolható számok nem fix számú tizedesjeggyel kerülnek tárolásra[4], hanem a karakterisztika alapján a mantisszában tárolt érték tizedespontja „mozog”.

A karakterisztika is lehet pozitív vagy negatív, de ebben az esetben nem az egész számoknál megismert előjeles (kettes komplemens) tárolást használjuk, hanem az *eltolt* [excess] tárolást: a tárolandó karakterisztikához hozzáadunk egy kellően nagy, fix értéket, és az így kapott eredményt tároljuk el, ami már biztosan pozitív egész szám.

[4] ezt a módot fixpontos ábrázolásnak nevezzük

1.3.13. példa. A $382_{[10]} = 101111110_{[2]}$ normalizált alakja: $1.01111110_{[2]} \cdot 2^{8_{[10]}} = 1.01111110_{[2]} \cdot 2^{1000_{[2]}}$, azaz tárolandó a 0 előjelbit, a 01111110 mantissza és az 1000 karakterisztika (a megfelelő eltolással).

1.3.14. példa. A $-3.375_{[10]} = -11.011_{[2]}$ normalizált alakja $-1.1011_{[2]} \cdot 2^1$, azaz tárolandó az 1 előjelbit, a 1011 mantissza és az 1 karakterisztika (a megfelelő eltolással).

1.3.15. példa. A mantissza 10 biten, az exponens 5 biten történő excess-15 ábrázolása esetén a tízezer ábrázolása: $10000 = 1.0011100010 \cdot 8192 = 1.0011100010 \cdot 2^{13}$, azaz tárolandó a 0 előjel, a 0011100010 mantissza és az 11100 exponens.

1.3.15. feladat. Hasonlítsuk össze a 8 bites előjel nélküli egész, az előjelbites egész, a kettes komplemes és a 127-tel eltolt (excess-127) számábrázolásokat! (Táblázatosan foglaljuk össze: egy sor legyen a tárolt 8 bit, az oszlopok legyenek a vizsgált ábrázolási módok, egy adott mezőbe írjuk be a mező sorának megfelelő bitsorozat értelmezését az oszlopnak megfelelő számábrázolás esetében!)

1.3.6. Lebegőpontos számábrázolás határai és pontossága

Az előjeles vagy előjel nélküli egész számok ábrázolásakor és a lebegőpontos számábrázolás esetében is csak fix értékek tárolhatók, de míg ezek az egészek esetében pontosan megegyeznek a tárolni kívánt egész számokkal, a lebegőpontos számábrázolás esetében ez nincs így, mivel végtelen sok valós szám nyilván nem ábrázolható véges helyen. Ezt úgy is értelmezhetjük, hogy a tárolás során kényszerű kerekítés történik. Mindezek miatt az ábrázolási határok mellett a pontosság is jellemez egy-egy konkrét lebegőpontos számábrázolást, ami megadja, hogy egy adott szám tárolása esetén a tárolni kívánt és a tárolt szám értéke legfeljebb milyen távol lehet egymástól. A lebegőpontos számok normalizált alakú tárolásából következik, hogy a (relatív) pontosságot a mantissza tárolási mérete határozza meg, az ábrázolási határok pedig elsődlegesen a karakterisztika méretéből adódnak. Fontos azt is kiemelni, hogy a pontosság az ábrázolási tartományban abszolút értelemben nem egyenletes, azaz függ az ábrázolni kívánt számtól.

1.3.16. példa. A mantissza 10 biten, az exponens 5 biten történő excess-15 ábrázolása esetén a tízezernél nagyobb, pontosan ábrázolható számok közül a legkisebb a 0 előjelű, a 0011100011 tárolt mantisszájú és a 11100 tárolt exponensű szám: $1.0011100011_{[2]} \cdot 2^{13} = 1.2216796875 \cdot 8192 = 10008$, a tízezernél kisebb, pontosan ábrázolható számok közül a legnagyobb a 0 előjelű, 0011100001 tárolt mantisszájú és a 11100 tárolt exponensű szám: $1.0011100001_{[2]} \cdot 2^{13} = 1.2197265625 \cdot 8192 = 9992$, azaz tízezer körül a hiba kevesebb, mint 8.

1.3.17. példa. Az IEEE binary16 számábrázolás esetén az egy tízezrednél kisebb, pontosan ábrázolható számok közül a legnagyobb a 0 előjelű, 1010001101 tárolt mantisszájú és 00001 tárolt exponensű szám: $1.1010001101 \cdot 2^{-14} = 1.6376953125 \cdot 0.00006103515625 = 0.00009995698929$, a tízezrednél nagyobb, pontosan ábrázolható számok közül a legkisebb a 0 előjelű, a 1010001110 tárolt mantisszájú és a 00001 tárolt exponensű szám: $1.1010001110 \cdot 2^{-14} = 1.638671875 \cdot 0.00006103515625 = 0.00010001659393$, azaz egy tízezred körül a hiba kevesebb, mint egy tízmilliomod.

1.3.16. feladat. A mantissza 10 biten, az exponens 5 biten történő ábrázolása esetén adjuk meg (az előző két példához hasonló módon) az ezernél, a száznál, a tíznél, az egy tizednél, az egy századnál és az egy ezrednél nagyobb számok közül a legkisebb ábrázolhatót illetve a kisebb számok közül a legnagyobb ábrázolhatót, és számítsuk ki (közelítőleg) a hibát.

A lebegőpontos számok ábrázolásának a gyakorlatban is alkalmazott nemzetközi szabványa a az IEEE 754 = IEC 599 = ISO/IEC 60559. Az ebben definiált konkrét bináris lebegőpontos ábrázolások közül néhány látható az 1.2. ábrán.

1.3.7. Alulcsordulás

A túlcsorduláshoz (ami pozitív vagy negatív irányban túl nagy szám ábrázolásának kísérletét jelenti, azaz a számábrázolási határt léptük túl) hasonló az alulcsordulás [underflow]: itt a számábrázolás másik paramétere, a pontosság szenved csorbát: olyan kis számot akarunk ábrázolni, ami már csak nullaként ábrázolható.

elnevezés	mantissza méret	karakterisztika méret	karakterisztika eltolás
binary16	10	5	15
binary32	23	8	127
binary64	52	11	1023
binary128	112	15	16383

1.2. ábra. IEEE 754 = IEC 599 = ISO/IEC 60559 bináris lebegőpontos típusok jellemzői

Az alulcsordulási hiba csökkentésére a szabványnak megfelelve a nulla exponens értéket speciálisan kell kezelni: a mantissza legnagyobb helyiértékű bitjét az előzőekben megismert fix 1 helyett nullának kell értelmezni, és az exponens excess értékéhez is egyet hozzá kell adni:

1.3.18. példa. Az IEEE binary16 formátumban tárolt 0000000000000001 bitminta értelmezése: előjel: 0, exponens: 00000 (az eredeti $0-15$ értelmezés helyett a módosított exponens: -14), mantissza: 0000000001, azaz: $0.0000000001 \cdot 2^{-14} = 2^{-24} = 5.9604645 \cdot 10^{-8}$

1.3.17. feladat. A mantissza 10 biten, az exponens 5 biten történő ábrázolása esetén adjuk meg a legnagyobb ábrázolható negatív számot.

1.3.8. Végtelenek és a NaN

A szabványos lebegőpontos számábrázolások a valós számokon kívül képesek tárolni a ∞-t és $-\infty$-t, továbbá a speciális NaN (Not a Number) értéket. Ez utóbbit kapjuk eredményül (többek között) akkor, ha nullát nullával osztunk vagy ha negatív számból vonunk négyzetgyököt.

Ha az exponens minden bitje 1 és a mantissza nulla, akkor az (előjeltől függően) $\pm\infty$ az ábrázolt érték, ha a mantissza nem nulla, akkor NaN az ábrázolt érték.

1.3.18. feladat. Adjuk meg a legnagyobb binary16-ban ábrázolható számot!

1.4. Karakterek és kódolásuk

1.4.1. Karakterek és karakterkészletek

A *karakter*t [character] (a számhoz hasonlóan) fogalomnak tekintjük, amit meg kell tudni jeleníteni írott formában, illetve el kell tudni tárolni a memóriában vagy bármely más tárolóeszközön, fájlban, illetve továbbítani kell tudni egy informatikai hálózaton. Karakter lehet az ABC egy betűje, egy szám, egy írásjel (a szóközt is beleértve) vagy egyéb más írásrendszerben használt jel illetve vezérlő karakter (például soremelés) is.

Karakterkészlet [character set, charset] alatt karakterek kiválasztott csoportját értjük (a kiválasztás lehet tetszőleges, de általában valamely ország, nemzetiség, nyelv, régió alapján történik).

1.4.2. Karakterek kódolása

Karakterek kódolása [character encoding, coded character set] alatt a karakterekhez valamilyen érték (kód) rendelését értjük. Ilyen például a Morze-kód, ami karakterekhez hosszabb és rövidebb impulzusokból álló kódokat rendel (amiket aztán könnyen lehet továbbítani például egy rádió adó-vevő segtségével), vagy a Braille-kód, ami karakterekhez 3D objektumokat rendel (amit aztán megfelelő technológiával „kinyomtatva” látásukban sérült emberek is képesek elolvasni). Az informatikában a karakterkódolás általában egy szám karakterhez rendelését jelenti (amit aztán valamilyen módszerrel tárolunk).

Karakterek tárolási formáján [characer encoding form & character encoding scheme] a karakter kódok (számok) konkrét tárolási módját értjük. Ez lehet triviális, például hogy egy megfelelő hosszúságú, egészek tárolására használt ábrázolást alkalmazunk, vagy lehet szofisztikáltabb, például egy tömörebb – de bonyolultabb – változó kódhosszúságú kódolás esetében.

Egyes kódolási módszerek egyszerre meghatározzák a karakterek kódolását és a kódok tárolási formáját. Tipikusan ilyen kódolási módszerek a klasszikus kódtáblák [code page, character map, charmap], amelyek általában egy bájton ábrázolnak egy karaktert.

1.4.3. Klasszikus kódtáblák

Az ASCII kódtábla

Az American Standard Code for Information Interchange (ASCII) 7-bites kódtábla látható az 1.3. ábrán. Az egyes karakterek kódja a karakter sorának és oszlopának fejlécéből adódik: például a @ kódja 0x40, a W kódja 0x57.

ASCII Code Chart

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

1.3. ábra. Az ASCII 7-bites kódtábla (forrás: wikipedia.org/wiki/ASCII)

Nagyon fontos kiemelni, hogy az ASCII kódtábla 7-bites, azaz 128 karakter kódolására alkalmas. Ha 8-biten tároljuk vagy továbbítjuk, a legnagyobb helyiértékű bitet nullára állítjuk.

1.4.1. feladat. A home1.paulschou.net/tools/xlate/ honlapon ellenőrizhetők az ASCII karakterek bináris átváltásai.

Az ISO 8859-X kódtáblák

Az ISO/IEC 8859-X kódtáblák az ISO és az IEC szabványosító testületek közös kódtáblái, céljuk, hogy minél több regionális karaktert tartalmazzanak. Az ASCII kódtábla az ISO 8859-X kódtáblák része, azaz minden ASCII kód egyben érvényes ISO 8859-X kód is. Mi a leggyakrabban az ISO 8859-1 (nyugat-európai) és az ISO 8859-2 (közép-európai) kódtáblákkal találkozhatunk, ezeket szokás latin-1 és latin-2 kódtábláknak is nevezni. Az ISO 8859-1 kódtáblát ISO 8859-15 (latin-9) néven frissítették (többek között belekerült az euro karakter), és hasonló történt az ISO 8859-2-vel is: ISO 8859-16 (latin-10) néven frissítették.

A Windows kódtáblák

Az ASCII kiterjesztésével a Microsoft megalkotta saját 8-bites kódtábláit, külön-külön egyes régiókra. Mi a leggyakrabban a windows-1250 (közép-európai) és a windows-1252 (nyugat-európai) kódkészletekkel találkozhatunk. Ezeket szokás windows latin-2 illetve windows latin-1 kódtábláknak is nevezni.

Szintén fontos tulajdonsága ezeknek a kódoknak is, hogy az ASCII-t módosítás nélkül tartalmazzák, azaz annak csak kiegészítései. Az ISO 8859-X kódtáblák és a windows-xxxx kódtáblák nagy részben egyeznek (nem kizárólag a közös ASCII részek), de nem teljes mértékben kompatibilisek egymással.

Feladatok

1.4.2. feladat. Megváltozik-e egy adott ASCII karakter kódja, ha előjeles vagy előjel nélküli egészként tároljuk?

1.4.3. feladat. Keressük meg az ISO 8859-1, az ISO 8859-2, a windows-1250 és a windows-1252 kódtáblák kiosztásait!

1.4.4. feladat. Helyes eredményt kapok, ha egy ASCII kódolt karakterekből álló fájlt a.) windows-1250 b.) windows-1252 kódtáblák alapján próbálom értelmezni?

1.4.5. feladat. Mi lehet a magyarázata annak, hogy régebben (sajnos sokszor még ma is) az ő illetve Ő betűk helyett (hibásan) õ vagy Õ szerepelt?

1.4.6. feladat. Tudunk-e több, különböző kódkészlethez tartozó karaktert egyetlen szövegfájlban tárolni (és azokat helyesen megjeleníteni olvasáskor)?

1.4.7. feladat. Adjunk példát olyan szövegfájlra, ami nem csak ASCII karaktereket tartalmaz, de mégis azonos módon értelmezhető a.) ISO 8859-2 és windows-1250 b.) ISO 8859-1 és windows-1252 kódtáblákkal!

1.4.4. A Unicode

Az előzőekben ismertetett kódtáblák közös problémája, hogy önmagukban nem képesek több nyelvű szövegek tárolására (sőt, egyes esetekben még egyetlen nyelv esetében sem, például: kínai, japán), mivel a – 8 bites tárolásból adódó – lehetséges 256 különböző karakter nyilván nem elegendő a világ összes nyelvében használt betű és írásjel ábrázolására. Ennek a megoldására jött létre a Unicode konzorcium, ami létrehozta és karbantartja a Unicode ajánlást. Jelenleg (Unicode 6.0) 93 írásrendszert és több, mint százezer karaktert tartalmaz, amik között élő és holt nyelvek mellett megtalálhatók matematikai és egyéb szimbólumok is.

1.4.8. feladat. A www.unicode.org/charts/ oldalon nézzük meg a Unicode által támogatott karaktereket!

A Unicode azonos az ISO/IEC 10646 szabvánnyal. A Unicode egy karakter kódolási szabvány (figyelem, önmagában nem kódtábla!), ami meghatározza, hogy egy konkrét karakternek mennyi a Unicode értéke [code point]. Általában U+hexa_kód formában jelöljük: például az euro jel kódja: U+20AC. Ezt a Unicode értéket különböző módokon lehet tárolni.

Az UTF-8

Az UTF-8 a Unicode egy tárolási formája (Unicode Transformation Format) ami a Unicode kódokat változó hosszon, 1-4 bájton tárolja, a kód értékétől függően. Legfontosabb tulajdonságai:

- ASCII kompatibilis, azaz minden ASCI szöveg egyben helyes UTF-8 szöveg is,
- önszinkronizáló, azaz nem kell az UTF-8 bájtsorozat elejéről kezdeni az olvasást, hogy pontosan el lehessen határolni az egyes karaktereket reprezentáló UTF-8 byte csoportokat.

Az UTF-16

Az UTF-16 tárolási forma változó hosszon, 2-4 bájton tárolja a Unicode kódokat. Nem ASCII kompatibilis.

Az UTF-32

Az UTF-32 tárolási forma fix hosszon, 4 bájton tárolja a Unicode kódokat. A kódolás nagyon egyszerű: az Unicode kódokat kell 4 bájtos egészekként tárolni. Nem ASCII kompatibilis.

Feladatok

1.4.9. feladat. Hogyan befolyásolják a tárolási méretet a használt karakterek? Mikor érdemes az UTF-8 és mikor az UTF-16 kódolási formát választanunk?

1.4.10. feladat. Az UTF-8 önszinkronizáló tulajdonságának milyen szerepe van a

- véletlenül kiválasztott pozícióból történő megjelenítésre,
- a hibás átvitelből adódó értelmezési problémákra?

1.4.11. feladat. Az UTF-8, UTF-16, UTF-32 kódolások közül melyek alkalmasak szövegek véletlenszerű elérésére (azaz például ha az x. karakterhez akarok közvetlenül ugrani)?

1.4.5. Szövegfájlok

Ha egy fájlban csak szöveget akarunk tárolni (pontosabban csak olyan karaktereket, amelyek mindegyike megtalálható egy kiválasztott karakterkészletben), akkor nincs más dolgunk, mint a karakterek (kódtáblája vagy valamely Unicode tárolási formája szerinti) bájtjait egymás után írni, és ezt eltárolni. Valójában is ez történik, ezeket a fájlokat nevezzük *egyszerű szövegfájl*oknak [plain text file], kiterjesztésük (általában): TXT.

1.4.6. Feladatok

1.4.12. feladat. Hasonlítsuk össze a 7 tárolási formáit: a.) előjel nélküli egészként, b.) kettes komplemens ábrázolású előjeles egészként, c.) ASCII kódolással d.) UTF-8 kódolással!

1.4.13. feladat. Honnét tudjuk, hogy egy szövegfájl beolvasásakor a kódokat melyik kódtábla szerint kell értelmeznünk?

1.4.14. feladat. Pusztán a szövegfájlba történő beleolvasással eldönthető-e általános esetben, hogy az milyen kódtábla szerint értelmezendő?

1.4.15. feladat. Töltsünk be a böngészőnkbe egy szövegfájlt, majd módosítsuk a kódtáblát! Kódoljuk át a szövegfájlt a iconv parancs segítségével, és nézzük meg az eredményt a böngészővel illetve az od programmal!

1.4.16. feladat. Töltsük be a www.itk.ppke.hu/oktatas oldalt, és a böngészőnkben állítsuk át a karakterkészletet! Mi történik?

1.4.17. feladat. Mi a mojibake? (Keressünk rá!)

1.4.18. feladat. Ha ékezetes karaktereket használunk egy SMS-ben, akkor van-e különbség az egy SMS-ben elküldhető karakterek száma között, ha magyar (jobbra dőlő) ékezetekkel rendelkező karaktereket vagy csak balra dőlő ékezetekkel rendelkező karaktereket használunk? Indokoljuk meg!

2. fejezet

Linux

Uhlár László

2.1. Bevezetés

Nagyon fontos, hogy a lentebb szereplő parancsokat kipróbáljuk! Erre több lehetőségünk adódik.

- van a saját számítógépünkön valamilyen telepített linux verzió és azt használjuk,
- letöltünk egy live cd-t és azt használjuk, pl.: PuppyLinux (http://skami.homelinux.org/)
- eddig nem volt, de szeretnénk a gépünkre linuxot is; egy kis leírás, bár nem a legfrissebb, de jól használható: http://hogyan.org/debian-5-lenny-telepites
- a putty programmal belépünk valamely egyetemi szerverre (users, turdus) és ott dolgozunk;

2.2. Egy kis történelem

2.2.1. A kezdetek

A számítógépek az ötvenes évektől a nyolcvanas évek elejéig a kutatók eszközei voltak és igazából a tömegek elől el voltak zárva. Több kutató használt egy nagy gépet, a fejlesztéseiket megosztották egymással, igazi kis közösségek jöttek így létre. A kor egyik legendás gépe volt a PDP10-es, ezen az ITS nevű operációs rendszer futott, ennek továbbfejlesztésén dolgozott Richard Stallman is (lásd később). A PDP10 fejlesztését azonban a gyártója a nyolcvanas évekre abbahagyta, az intézeteknek újabb gépek után kellett nézniük. Ezek már más operációs rendszert futtattak, melyek nem voltak szabadok, már ahhoz is titoktartási szerződést kellett aláírniuk, ha egy futtatható másolatot akartak. Azaz tilos lett egymásnak segíteni, az eddig együttműködő közösségek felbomlottak, nem oszthatták meg egymással fejlesztéseiket.

2.2.2. A GNU

Richard Stallmann ezt az új helyzetet nem tudta elfogadni, elhatározta egy új, teljesen nyílt operációs rendszernek a megírását, 1983 táján létrehozta a GNU projektet, hogy terveit megvalósítsa. (A GNU jelentése: GNU is not UNIX). Ekkor fogalmazta meg a GNU kiáltványt, melyet teljes terjedelmében például a gnu.hu oldalon olvashatunk el.
Ekkor fogalmazta meg a szabad szoftverekkel kapcsolatos alapelveit:

- a program szabadon használható bármilyen célra.
- a programot bárki szabadon módosíthatja igényei szerint.
- a programot bárki továbbadhatja akár ingyen akár pénzért.
- a program módosított verziói szintén szabadon terjeszthetők.

Ezen elveket a jogászok számára is elfogadható formába kellett önteni, így jött létre egy különleges licenc, a GPL (General Public Licence). Teljes szövege magyarul szintén a gnu.hu oldalon olvasható. Ekkoriban jött létre a szabad szoftvereket támogató alapítvány, az FSF (Free Software Foundation). Céljukról, működésükről többet olvashatunk pl.: az fsf.hu oldalon.

Már csak egy valami hiányzott: egy olyan kernel (az operációs rendszer magja), melyen futtatni lehetne a GNU programokat.

2.2.3. Linus Torvalds

1991-ben egy finn egyetemista, Linus Torvalds épp egy új projekten kezdett el dolgozni, egy új, szabad operációs rendszeren, melyben ki akarta javítani az oktatásra akkoriban előszeretettel használt MINIX operációs rendszer hibáit, hiányosságait. Azaz adott volt egy kernel (Linus munkája) alkalmazások nélkül és adott volt egy alkalmazás gyűjtemény (GNU) kernel nélkül. Nem kellett sok idő ahhoz, hogy egymásra találjanak, így született meg a Linux, amit helyesen GNU/Linux-nak kellene neveznünk. Pár év, és megjelentek az első disztribúciók: a kernel és a rengeteg GNU alkalmazás közül néhány összeépítve egy jól használható rendszerré. (pl.: Debian: 1993. augusztus 16.)

2.3. Disztribúciók

A Wikipédiából, a szabad enciklopédiából:
A Linux-disztribúció (röviden: distro vagy disztró) olyan összeállítás, mely egy felhasználásra kész Linux vagy GNU/Linux alapú operációs rendszert, és ahhoz tartozó, válogatott programokat tartalmaz.

Operációs rendszerekkel kapcsolatban a disztribúció szó alatt egy gondosan egybeválogatott, általában rendszermagból és felhasználói programok csomagjaiból álló terjesztést értünk.

2.3.1. Disztribúciók közötti leggyakoribb különbségek

Disztribúciókat legtöbbször az különbözteti meg, hogy milyen célközönségnek és milyen feladatra készítik őket, így mindenki megtalálhatja a neki leginkább megfelelőt. Így léteznek olyanok, melyek lehetőséget nyújtanak arra, hogy szinte az összes konfigurálási lehetőséget egy grafikus felületen végezzük el és vannak olyanok is, amelyek megkövetelik, hogy a felhasználó mindent a konfigurációs állományok szerkesztésével állítson be a saját ízlésének megfelelően. Egyes disztribúciók célja, hogy mindig a lehető legfrissebb szoftvereket szállítsa, míg mások jól kitesztelt, stabil, ám emiatt kissé elavult csomagokat szállítanak. A legtöbb disztró adott közönséget céloz meg: profi vagy kezdő felhasználókat, adminisztrátorokat, „buherátorokat”, kevés memóriával rendelkező vagy csak CD-t tartalmazó gépeket stb. Néhány disztró a grafikus környezetet, míg mások inkább a karakteres konzolt támogatják.

További fontos különbség, hogy milyen csomagkezelőt használnak az adott terjesztésben. A könyvtárstruktúra általában hasonló módon van felépítve, viszont kisebb különbségek adódhatnak e tekintetben is, extrém esetekben teljesen eltérő felépítést is alkalmaznak a disztribútorok (pl.: GoboLinux). A disztrók egyik fő jellemzője az egyes programcsomagok installálásának, eltávolításának és frissítésének megkönnyítése és támogatása (lásd még: APT, RPM). A csomagkezelők a rengeteg feltelepíthető program karbantartását, frissítését, telepítését, stb. teszik könnyebbé: például a GNU/Debian 6 esetében majd 30000 különböző program közül válogathatunk, így szinte biztosan megtaláljuk a felmerült feladataink megoldásához szükséges szofvereket e bőséges választékban. Az egyes programok, csomagok pontos verzió számmal vannak ellátva, egy-egy program megfelelő működéséhez szükség lehet más programokra is, azaz függőségei lehetnek. Ezen függőségek (lehetőleg automatikus) feltelepítését is a csomaglezelők végzik.

Hardvertámogatás terén is adódhatnak különbségek, viszont alapvetően mind ugyanazt a kernelt használják, így elviekben ha egy disztribúció alatt egy hardver működik, akkor az bármely más, az adott architektúrát támogató disztribúció alatt is működésre bírható. Vannak céldisztribúciók is, például kifejezetten tűzfal vagy router üzemelésére. Megkülönböztethetjük őket az alapján is, hogy server, desktop vagy embedded felhasználásra szánják.

A disztrók nagy részének készítői komolyan veszik a biztonsági problémákat, és az ismert hibák javításait rövid időn belül elérhetővé teszik disztrójuk csomagfrissítési módszerének segítségével.

Nagy eltérések vannak a disztrók kiadásai között eltelt időben; egyes disztrók fix ciklust alkalmaznak (például 6 hónaponként egy új kiadás), más disztróknál nincs kötött kiadási ciklus. Léteznek kereskedelmi terjesztések és vállalati és otthoni/kisirodai disztribúciók is.

Nem mindegyik disztró ugyanazt a kernel verziót használja, továbbá sok disztró saját igényeinek megfelelően módosítja a hivatalosan kiadott, ún. „vanilla kernelt."

A nagyobb és ismertebb disztribúciók (a teljesség igénye nélkül):

- blackPanther OS, magyar Linux-disztribúció
- UHU-Linux, magyar Linux-disztribúció
- Debian GNU/Linux
- Ubuntu, Kubuntu, Xubuntu
- Mandriva
- PCLinuxOS
- Red Hat Linux
- Fedora
- CentOS
- openSUSE
- Slackware
- Gentoo
- Arch Linux
- Knoppix, Damn Small Linux, Live CD-ként való futtatásra tervezve
- CrunchBang Linux

Egy kis érdekesség: http://upload.wikimedia.org/wikipedia/commons/8/8c/Gldt.svg

2.4. Könyvtárszerkezet

Linux alatt egy „tökéletes" fa struktúrába van szervezve a teljes könyvtárszerkezet, (azaz ne számítsunk C, D, ... meghajtókra!) Mindennek az alapja a `/` jellel jelölt gyökérkönyvtár, más néven root. Ez minden Linux alapja, ebből ágaztatható le a teljes szerkezet.

2.4.1. példa. Adjuk ki a következő utasítást: `ls /` Hasonló listát kell látnunk:

```
bin
boot
cdrom
dev
etc
home
lib
lost+found
media
mnt
opt
proc
```

```
root
sbin
sys
tmp
usr
var
vmlinuz
```

Ezek a főkönytárak majdnem minden Linuxban változatlanul megvannak, leszámítva talán a /cdrom-ot és /media-t. A /media egy újabb „találmány”, ide kerülnek a cserélhető médiák. Nézzük, melyikben mi található:

bin, sbin: A bin könyvtárakban futtatható állományok vannak. Több bin könyvtár is található ezen kívül, például a /usr/bin és a /usr/sbin. Bár ez nem törvényszerű, de általában a bin könyvtárakban a minden felhasználó által elérhető programok kerülnek az sbin könyvtárakba pedig olyan rendszereszközök, melyeket általában rendszergazdák használnak. A /bin és /sbin az alaprendszerhez, a boot folyamathoz szükséges programokat tartalmazza, a felhasználói programok a /usr/bin /usr/sbin alá kerülnek.

boot: a boot könyvtárban találhatók a bootnál fontos fájlok: általában a rendszermag (kernel), illetve Grub rendszerbetöltő esetén annak konfigurációs állománya is.

cdrom:Ez alá csatolódik be a CD meghajtó egység.

dev: Linux alatt fájlokon keresztül érünk el mindent, a CD-vel kezdve, a hangon át, az egérig. Ezek a speciális eszközfájlok találhatók ebben a mappában.

etc: Az etc könyvtár a gyűjtőhelye a különböző programok globális konfigurációs fájljainak. Ellentétben a Windowsos registry megoldással, Linux alatt minden konfigurációs állomány egyszerű szövegfájlba van mentve, aminek nagy előnye, hogy az állományok akkor is egyszerűen elérhetők, ha a rendszer egyébként használhatatlan. Természetesen emellett az egyes programok felhasználó specifikus beállításokkal is rendelkeznek, ezeket a home könyvtárakban tárolja a rendszer, rejtett mappákban.

home: ez alatt a könyvtár alatt találhatók a felhasználói könyvtárak, az adott könyvtár alatt a felhasználónak teljes dúlási joga van, ezen az egy könyvtáron kívül azonban leginkább csak olvasási joga van alapból.

lib: a lib könyvtár alatt már a rendszer részei lapulnak: library fájlok, kernel modulok, stb.

lost+found: egy speciális könyvtár, jelen esetben egy ext3 típusú fájlrendszerrel szerelt partícióról van szó, ez a könyvtár nem is a Linux, mint inkább a fájlrendszer része.

media: rendszerfüggő a dolog, általában a /media könyvtár alá kerülnek befűzésre a CD/DVD eszközök, pendirve illetve a floppy. Röviden: a cserélhető médiák.

mnt: a másik „betűzögetős” könyvtár. Ez alá a könyvtár alá kerülnek (általában) befűzésre a fix partíciók. Mivel ebben a könyvtárstruktúrában nincs kiemelt „volume” egy egy meghajtónak, mint Windows alatt a C:, D:, stb., így egy-egy eszközt tetszőleges helyre befűzhetünk a fájlrendszerbe. Különösen praktikus ez például home könyvtár estén: ha kinőjük az e célra fenntartott partíciót, és veszünk egy új vincsesztert, egyszerűen csak rámásoljuk anyagainkat, letöröljük az eredeti példányt, majd befűzzük a /home könyvtár alá az új adathordozót.

opt: a hivatalos leírás szerint külsős programok települnek ebbe a könyvtárba, de a rendszerek nagy részén üresen áll...

proc: Itt találhatóak az éppenfutó műveletek -fájlként leképezve, sorszámozva, illetve információk a rendszerről: processzorról, memóriáról, stb. Nagy mennyiségű hasznos információt talál itt az avatott kéz.

root: A rendszergazda (root) felhasználói könyvtára

tmp: Az egyes programoknak szükségük van/lehet átmeneti fájlokra. Ezek kerülnek ide. Ez a másik olyan könyvtár, amely alapértelmezetten írható minden felhasználó számára.

usr: Ez alatt a könyvtár alatt található minden. Persze ez így kicsit túlzónak hat, de majdnem igaz: az usr könyvtár alatt található a telepített programok nagy része, hagyományból ide szokott az ember fia forrásokat pakolni (/usr/src), és azt leforgatni. Itt találhatók a dokumentációk, itt találhatók az ikonok nagy része, sorolhatnánk a végtelenségig...

var: Szintén számos szolgáltatás gyűjtőkönyvtára. Itt találhatók a naplófájlok, egyes programok hosszabb ideig tárolt, mégis átmeneti fájljai, alapértelmezettben a felhasználói levélboxok, stb.

2.4.2. példa. Nézzünk bele az egyes könyvtárakba: adjuk ki a következő utasítást (utánuk ENTER): `ls /bin` (aztán `ls /boot`, `ls /home`,...)

2.4.3. példa. Gépeljük be, majd nyomjunk ENTER-t: `cat /proc/meminfo`

Az ls parancs kilistázza az adott könyvtár bejegyzéseit.

2.5. Jogosultságok

A linux disztribúciókban található egy kitüntetett felhasználó, a rendszergazda, a telepítés során jön létre, a neve: root. Neki mindenhez joga van, bármit törölhet, bármit megnyithat, létrehozhat felhasználót, stb. Az ő általa indított programok az ő jogaival futnak, egy szándékosan vagy véletlenül megváltoztatott program a root jogaival futva komoly károkat tud okozni. Ezért a legtöbb disztribúcióban létre kell hozni már a telepítéskor egy korlátozott jogú felhasználót, akinek az adataival belépve korlátozott jogokkal tudunk dolgozni. Ez így biztonságos!

2.5.1. példa. Írjuk be: `cat /etc/passwd` A kapott hosszú lista első oszlopa a rendszerünkön lévő felhasználók neveit tartalmazza, a sajátunkat is ott kell látnunk. (Talán kiderült már: a cat utasítással szöveges fájlok tartalmát lehet kilistázni.)

A lentebb bemutatásra kerülő jogoknak igazi jelentősége a több felhasználó által használt rendszerek esetében van (pl.: users és turdus szerverek), ha egy gépet csak egyedül mi használunk, a jogosultságok állítgatása nem lesz annyira fontos.

Minden felhasználó valamilyen csoportnak is tagja (akár többnek is), mindenkinek van egy alapértelmezett csoportja (elsődleges csoport), ez Debian rendszeren megegyezik a felhasználó nevével, a felhasználó létrehozásakor jön létre, az új felhasználó egyből belekerül.

2.5.2. példa. Adjuk ki a következő utasítást: `cat /etc/group` A kapott lista első oszlopa a rendszerünkön lévő csoportok neveit tartalmazza.

A Linux fájlrendszere tárolja a fájl tulajdonosának azonosítóját a fájlhoz tartozó csoportot és a hozzáférési jogosultságot is. A hozzáférési jogosultságok ábrázolásához egy három részből álló kódot használ, amit fájlmodnak nevezünk.

- Első rész a saját (user) jogot
- Második rész a csoport (group) jogot
- Harmadik rész mindenki más (others) jogait rögzíti

A saját jog alatt a fájl tulajdonosának jogait értjük, legtöbb esetben ő az adott fájl vagy könyvtár létrehozója is. Mindegyik rész a következő komponensekből áll:

- r (Read): olvasási jog (vagyis az adott fájl ezáltal olvasható)
- w (Write): írási jog (az adott fájl ezáltal válik írhatóvá)
- x (eXecutable): végrehajtási jog (futtatási jog)

2.5.1. Jogosultságok megváltoztatása

Egy fájl tulajdonosi (hozzáférési) jogait csak a fájl tulajdonosa, vagy a rendszergazda tudja megváltoztatni a következő paranccsal: `chmod +|-<mód> <fájlnév>` Meg kell határozni az alábbiakat: adunk vagy elveszünk jogot (+: adunk, -: elveszünk), kinek/kitől (saját, csoport, mindenki más (ugo)), milyen jogot adunk (r w x / 4 2 1).

2.5.3. példa. Saját magunknak írási jog: `chmod u+w munka.tar.gz`

2.5.4. példa. Másoknak futtatási jog: `chmod o+x munka.tar.gz`

2.5.5. példa. Egyszerre több jogot is meg lehet változtatni: `chmod o+x,u+w munka.tar.gz`

2.5.6. példa. Mindenkinek minden jog: `chmod 777 munka.tar.gz` ugyanezt a funkciót valósítja meg a `chmod a+rwx munka.tar.gz`

2.5.7. példa. Csak nekem legyen jogom mindenhez: `chmod 700 munka.tar.gz` ugyanaz mint: `chmod u+rwx,g-rwx,o-rwx munka.tar.gz`

Fájlok esetében a végrehajtási jognak csak a futtatható fájloknál van jelentőségük (bináris állományok, scriptek). Könyvtárak esetén az olvasási jog azt jelenti, hogy elolvashatja a fájlok neveit az adott könyvtárban, az írási jog jelenti, hogy a könyvtárban állományt, könyvtárat hozhatunk létre, míg a futtatási jog megengedi a belépést a könyvtárba.

2.5.2. Alapértelmezett jogok

Amikor egy fájlt létrehozunk, akkor az a jogosultságoknak alapértelmezett értékével fog rendelkezni. Pl.: Létrehozunk egy üres fájlt:

```
$ touch akarmi
$ ls -la akarmi
-rw-r--r--  1 bnorbert  staff  0 Okt 31 06:14 akarmi
```

A létrehozáson kívül, alapértelmezés szerint írási és olvasási joggal, a csoportba tartozók és mindenki más pedig csak olvasási joggal rendelkeznek. Ennek az az oka, hogy az operációs rendszer a fájl létrehozásakor a 022 maszkot alkalmazza. Egy állomány létrehozásakor alapértelmezésben senki sem kap futtatási jogot. Az alapértelmezett maszk lekérdezhető a következő paranccsal:

```
$ umask
022
```

Könyvtárak létrehozása esetén a 777-ből vonódik ki a mask, azaz alapértelmezetten egy könyvtár 755 jogokkal jön létre. Fájloknál a 666-ból vonódik ki a mask, így 644 jogokkal jönnek létre a fájljaink.

2.6. Parancsok

Néhány gyakran használt, fontosabb parancs:

man: a manual szóból, a parancsainkról kaphatunk információt. Írjuk be például: `man cat`

pwd: print working directory, kiírja az aktuális könyvtár teljes nevét (elérési útját). Próbáljuk ki: `pwd`

mkdir: make directory, könyvtárat hoz létr. pl.: `mkdir proba`

rmdir: remove directory, alkönyvtárat töröl. pl.: `rmdir proba`

cd: change directory, könyvtár váltás. pl.: `cd Desktop` (ha van grafikus felületünk).

ls: list, kilistázza az adott könyvtár bejegyzéseit. pl.: `ls /`

cp: copy, fájlok másolására használható: cp mit hova (további infók: man cp).

mv: move, fájlok mozgatása.

rm: remove, fájlok törlése.

cat: fájlok tartalmának megjelenítése.

wc: word count, a bemenetére küldött szövegben különféle statisztikákat számol: sorok száma, szavak száma, karakterek száma, ...

echo: az adott szöveget kiírja a standard kimenetre. pl.: `echo szöveg`

grep: a bemenetére érkező szövegből azokat a sorokat adja vissza, melyek egy megadott mintát tartalmaznak.

ln: hard link illetve soft link létrehozása. Egy adott állományra több helyről több néven hivatkozhatunk anélkül, hogy arról másolatot kellene készítenünk.

ps: processes, kilistázza a futó processzeket (folyamatokat)

& : háttérben indítja az adott programot. pl.: `top &`

bg: background, az adott, előtérben futó programot háttérbe küldi.

fg: foreground, előtérben futtatja az eddig háttérben futó programot.

kill: megállítja az adott folyamatot, programot.

date: kiírja az aktuális dátumot.

df: disk free, egy kis statisztikát jelenít meg az egyes partíciók foglaltságáról. pl.: `df -h`

du: disk usage, az egyes állományok, könyvtárak méretéről készít kis statisztikát. pl.: `du -hs ./` (a man alapján próbáljuk meg értelmezni az egyes kapcsolókat, paramétereket!)

ncal: calendar, egy kis naptár program. pl.: `ncal 2011`

Természetesen a listát még hosszasan lehetne sorolni, aki további parancsokkal szeretne megismerkedni, használja ki az internet lehetőségeit! Bármely kereső a „linux parancsok” kifejezésre több jól használható oldalt is ajánl.

2.7. Átirányítás

Bejelentkezés után egy program várja az utasításaidat, hogy azokat végrehajtsa: ez a shell. Több fajtája van, pl.: sh, bash, csh ... A számunkra szükséges szinten mindegyik egyformán működik. A bejelentkezés után a villogó prompt jelzi, hogy a shell várja az utasításainkat. A fentebb bemutatott parancsok után nézzünk valami különlegességet, amivel egész összetett feladatokat is megpróbálhatunk végrehajtatni a shellel: ez az átirányítás.

Egy programnak induláskor egy bemenete (stdin – szabványos bemenet, általában a billentyűzet) és két kimenete (stdout – szabványos kimenet, általában a képernyő, stderr – szabványos hibakimenet, szintén általában a képernyő) van. Lehetőség van bármelyik átirányítására fájlba illetve fájlból:

2.7.1. példa. Szeretném tudni, hogy a gépemen mely felhasználó home könyvtára mennyi helyet foglal. Ehhez kiadhatom a `du -hs /home/*` parancsot.

Ez a képernyőre listázza az eredményt, amit kértem. Igen ám, de lehet, hogy ezt szeretném megőrizni, hogy meg tudjam mutatni az érintetteknek, esetleg szeretném magamnak automatikusan, időről időre elküldeni e-mailben. A parancs kimenetét átirányítjuk egy fájlba, így nem vész el.

2.7.2. példa. Gépeljük be: `du -hs /home/* > foglalas.dat`

Ez a parancs szabványos kimenetét átirányította a foglalas.dat fájlba. Ha ez eddig nem létezett, akkor létrehozza a shell, ha létezett, akkor a tartalmát kitörli és csak az új tartalom lesz benne. Ha nem szeretnénk kitörölni a már létező fájlunk már létező tartalmát, akkor a következőképpen járhatunk el:

2.7.3. példa. `du -hs /home/* >> foglalas.dat`

Figyeljük meg a két átirányítási mód közötti különbséget. Igen ám, de ha a fenti parancsot kiadjuk, akkor egy sor hibaüzenet is keletkezhet, hiszen a legtöbb felhasználó home könyvtárába nincs jogunk „belekukkantani". Lehetőségünk van arra, hogy a szabványos hibakimenetet is átirányítsuk egy fájlba, ami ekkor a monitoron nem jelenik meg:

2.7.4. példa. Írjuk be: `du -hs /home/* > foglalas.dat 2>foglalas.hiba`

Igen, a 2 szám a hibacsatorna azonosítója, a 2>foglalas.hiba utasítással irányítottuk a hibaüzeneteket egy másik fájlba. Ha nincs szükségünk a hibákra a fájlban és a képernyőn sem akarjuk látni, akkor egy különleges fájlba, a semmibe is átirányíthatjuk a hibakimenetet:

2.7.5. példa. `du -hs /home/* 2> /dev/null`

A bemenet átirányítása annyit jelent, hogy a programom a bemenetét nem a billentyűzetről kapja, hanem egy fájlból. Egyszerű példa: az előzőleg elkészített foglalas.dat fájlból keressük meg azon sorokat, melyekben gigabájtnyi helyfoglalásról van adat:

2.7.6. példa. `grep < foglalas.dat G`

Bizonyos mintát (G) tartalmazó sorokat keres a grep (man grep) utasítás, de ebben az esetben a bemenetét a megadott fájlból veszi.

Újabb hasznos lehetőség a csövezés (pipe), amikor az egyik parancs kimenete alkotja a másik bemenetét. Ennek jele a |. Pl.: irassuk ki a passwd állomány sorait abc szerint rendezve:

2.7.7. példa. `cat /etc/passwd |sort`

A cat program kilistázza a fájl tartalmát, ez átadódik a sort program bemenetére (man sort).

2.8. Feladatok

2.8.1. feladat. Nézz utána, hogy mit csinál az ncal parancs!

2.8.2. feladat. A hét milyen napján születtél?

2.8.3. feladat. Mekkora helyet foglalsz a users.itk.ppke.hu szerveren?

2.8.4. feladat. Hozd létre a következő könyvtárstruktúrát a saját könyvtáradon belül!

```
./szulok/apa
./szulok/anya
```

2.8.5. feladat. Hozz létre egy fájlt (akár üreset is lehet) az apa alkönyvtáron belül! (touch, esetleg nano, esetleg cat,...)

2.8.6. feladat. Másold át az anya alkönytárba!

2.8.7. feladat. Írasd ki egy fájlba az elmúlt 10 percben módosított fálok neveit a munkakönyvtáradon belül! (find parancs)

2.8.8. feladat. Fűzd hozzá a fájl végéhez az aktuális dátumot! (date és átirányítás)

2.8.9. feladat. Módosítsd az előző fájl jogait, hogy neked csak írási jogod, másoknak (csoport, egyéb) pedig semmilyen joga ne legyen!

2.8.10. feladat. Próbáld meg a tartalmát kilistázni! (pl.: cat)

2.8.11. feladat. Szerezz információkat az od programról! (man, keresők,...)

2.8.12. feladat. Add ki a következő utasítást:
verb=echo ő | od -t x1
Értelmezd az eredményt!

3. fejezet

Adattárolás és Perifériák

Tuza Zoltán

3.1. Bevezetés

A számítógép futása során az egyes számításokból (rész) eredmények keletkeznek, amit a korai számítógép-generációktól kezdve már jelen lévő illékony[1] memória tárolt el. Később született meg az igény arra, hogy ezeket az eredményeket eltároljuk a számítógép két bekapcsolása között, egy program két futtatása között. Szükséges tehát, hogy tároláshoz ne legyen szükségünk elektromos áramra. Erre kezdetben nyomtatót, illetve lyukkártyákat használtak (ez utóbbi annyival volt szerencsésebb, hogy egy lyukkártya-olvasóval könnyen vissza lehet tölteni az információt a memóriába). Később megjelentek a szalagos tárolási módszerek, melyek segítségével nagy mennyiségű adatot tudtunk lineárisan elmenteni (emlékezzünk a magnókazettákra, ahol ha egy számot ki szerettünk volna hagyni, azt a szalag gyors tekeréssel tudtuk csak átlépni). Az áttörést a cserélhető lemezes olvasó jelentette, ahol egy mágnesezhető korongot forgattunk egy mozgatható mágnenes olvasó/író fej előtt, így szemben a mágneses szalaggal, ahol az a olvasó fej fixen volt tartva. Ezzel a fejet különböző a korong különböző részein lévő adatsávok fölé tudtuk helyezni. Ezt a módszert - melyben a tárolón lévő adatok bármelyikét a többi adat érintése/átlépése nélkül érhetjük el - hívjuk véletlen hozzáférésnek (Random Access). Ezen a ponton két irány indult el, az egyik mentén a cserélhető lemezek fejlesztették, még a másik vonalon létrejöttek fix- vagy merevlemezek. Mivel ezek az eszközök nem közvetlenül vannak a számítógép alaplapjába építve, hanem valamilyen csatlakozón keresztül kapcsolódnak hozzá, így adattároló perifériák gyűjtőnévvel hivatkozunk rájuk. Például az egeret és a monitort is perifériának tekintjük, az egyiket adatbeviteli perfiériának, még a másikat adatmegjelenítő perifériának hívjuk. Három legfontosabb fontos jellemzője egy adattároló eszköznek a következőek:

- Hozzáférési idő (seek time): az az idő ami az adat megcímzése és az adat kiolvasása között eltelik.
- Adatátviteli sebesség: egy időegység alatt hány byte információt tudunk az eszközre írni, vagy arról olvasni. Ez a jellemző alapvetően két elemből tevődik össze: az eszköz hozzáférési ideje és a csatolófelület átviteli sebessége (például: SATA 3.0 szabvány: max. 6 Gbit/s, USB 3.0 szabvány: max. 5 Gbit/s).
- Tárolókapacitás: hány bit információt tudunk eltárolni rajta. Fontos tisztázni, hogy ez egy bruttó érték, mivel nem a "nyers" winchesztert használjuk a nettó érték ettől eltérő lehet hiszen a különböző logikai fájlrendszerek más-más módon osztják fel a merevlemezt - Lásd a fájlrendszerek részt.

További fontos jellemző még, az adattárolás élettartalma, azaz meddig képes egy eszköz a ráírt információt megőrizni. A mágneses elven működő eszközök általában előbb szenvednek mechanikai hibából kifolyó adatvesztést, mint hogy a mágnesen elven tárolt adat elveszne. További, külső

[1] tápfesztültség megszünésével a bennetárolt információ elveszik

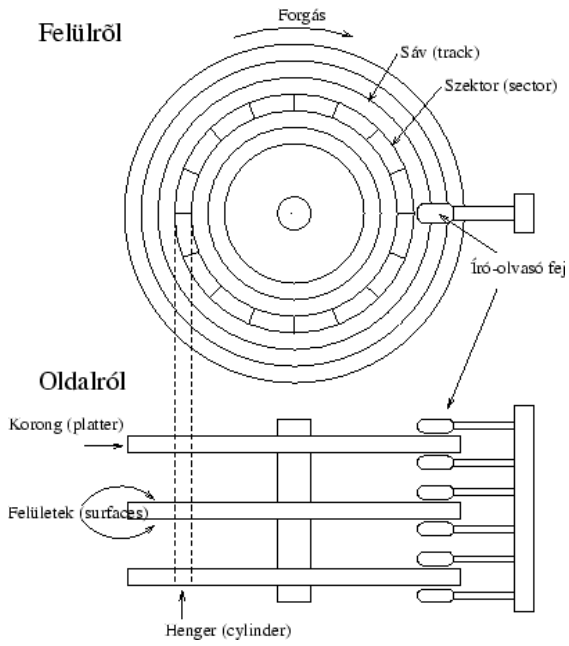
tényezők miatt is sérülhet az adatintegritás, például mechanikai behatás vagy hőhatás[2]. Míg a Compact Diskek (CD) esetében például a felület elgombásodása jelent veszélyt az információra nézve. Az Információ- és kódelmélet c. tárgyban részletesen tárgyalásra kerülnek az információ tömörítéséhez illetve az adatvesztéssel szembeni részleges rezisztenciához szükséges módszerek.

3.2. Adattároló perifériák

3.2.1. Merevlemez (Hard Disk Drive - HDD) felépítése

Ahogyan a cserélhető lemezes adattárolást, úgy a "fix" vagy "merevlemezes" adattárolást is az IBM mérnökei találták az 1950-es években[3]. Maga az adattárolás elve az elmúlt ötven évben nem sokat változott, egyedül a megbízhatóság és a tárolható adatmennyiség mértéke nőtt drasztikusan. A légmentesen vagy porszűrővel lezárt ház a következő részeket tartalmazza:

- mindkét oldalán mágnes anyagot tartalmazó korongok
- olvasó/író fejek, amelyek a mágnes felület felett pár nanométerrel - légpárnán siklanak.
- vezérlő elektronika, amely pozicionálja a fejeket, olvasás esetén elvégzi az analóg mágneses mérés digitalizálását, illetve a csatoló felületnek megfelelő adatátviteli protokollt megvalósítja.
- forgatómotor, jellemzően 5400 illetve 7200 forulat/perc sebességgel forgatja a lemezeket, tehát az olvasó fejek - átmérőtől függően - kb. 270 km/h-val száguld a lemezek felett.



3.1. ábra. A merevlemez vázlatos képe

A lemezeket a különböző kerületek mentén sávokra osztják, a sávokat pedig szektorokra, ez a legkisebb címezhető egység egy merevlemezen ezek mérete régebben 512 byte volt, jelenleg elérhető 4096 byte is. Mivel az olvasófejek együtt mozognak, ezért a különböző lemezeken azonos sávon állnak minden idő pillanatban, ezeket a sávokat együttesen cilindernek nevezzük. Látható tehát, hogy az összetartozó adatokat szomszédos szektorokra ill. azonos cilinderekre érdemes

[2]Minden mágneses anyagnak létezik egy úgynevezett Curie-pontja, ezen hőmérséklet felett az anyag elveszti mágneses tulajdonságát

[3]Az első merevlemezt IBM 350 RAMAC néven forgalmazták és 5 megabyte tárolókapacítással rendelkezett, ezt ötven darab 24"-os lemezzel érték el

írni. Fontos fogalom még a klaszter, ami az azonos sávon egymásután elhelyezkedő szektorok gyűjtő neve. Lásd a 3.1. ábrán. A merevlemez hatékony felosztása és az adatok tárolása a merevlemezre telepített fájlrendszer feladata, melyet részletesen tárgyalunk.

3.2.2. Compact Disk (CD) ROM felépítése

A cserélhető lemezes fejlődési vonalat az utóbbi évtizedig a mágneses tárolási elven működő eszközök határozták meg, de ezek tárolókapacitása nem nőtt és/vagy hozzáférési ideje nem csökkent olyan mértékben, mint a Compact Diskeké, ezért a továbbiakban nem is foglalkozunk velük. A CD-s adattárolás egy - a merevlemezes tárolástól eltérően - egy alapvetően optikai elven működő tárolási módszer. Egy lézerdióda által kibocsátott koherens fénysugár tapogatja le a CD lemez felületét, melyen apró gödrök és púpok váltják egymást, melyekről másként verődik vissza a lézersugár - ezzel reprezentálva a bitek értékeit. Gyártás során a CD lemezen - mint a merevlemeznél - sávokat és szektorokat hoznak létre, amelyben a biteket a vájatok reprezenetálják. Pl.: ha van mélyedés akkor az logikai egyes jelent, ha nincs akkor logikai nullát. Amikor az olvasó fej mindig azonos szögből megvilágítja a felületet, akkor a vájatba beeső lézer fény máshova verődik vissza, mint az lézer fény, mint ami nem esett bele a vájatba. Mivel a lemez fixen síkban forog és az olvasófej is meghatározott szögben világítja meg a felszínt, ezért a várható visszaverődési helyekre fényérzékeny szenzorokat helyeznek el. Értelemszerűen tehát a logikai egyes és nullás értékek más-más szenzorból váltanak ki jelet. Ebből a felépítésből látható, hogy minél fókuszáltabb a lézersugár, illetve minél kisebbek a vájatok a lemez felületén, annál nagyobb az elérhető adatsűrűség (természetesen további fontos paraméterek is vannak - anyagtechnikai jellemzők, a lézer hullámhossza, valamint több adattároló réteggel rendelkező lemezek is léteznek). CD ROM elnevezésben a ROM (Read Only Memory/Media) rövidítés arra utal, hogy ezeket az eszközöket, csak egyszer lehet ír, utána már csak olvashatóak, léteznek újra írható CD lemezek is, ahol egy az olvasástól eltéri tulajdonságú lézersugárral visszállítják az eredeti felületet - természetesen ebben az esetben nem vájatokkal dolgozunk, hanem a felülnek változtatjuk meg a visszaverődési tulajdonságait az írás során.

3.2.3. Pendrive, Flashdrive

Mivel ezen eszközök működésének megértéséhez komoly elektronikai háttér tudásra van szükség, ezért a technikai részleteket nem tárgyaljuk. A működési elvről elegendő annyit megjegyeznünk, hogy ezek a tárolók olyan különleges áramkörök, amelyekben lévő tranzisztorok tápfeszültség jelenléte nélkül is képesek megtartani azt az állapotot, amit tápfeszültség jelenlétében megadtunk nekik, ezzel reprezentálva a logikai 1-0 értékeket. Fontos még megjegyezni, hogy ebből a technikai megvalósításból kifolyóan az ilyen típusú eszközök írási és olvasási sebessége jelentősen eltér egymástól. Az érdeklődők a következő kulcs szavak mentén tudnak további információhoz jutni: Flash memory, Floating gate, EEPROM.

3.3. Adatátviteli technológiák

A programok futtatásához az adattároló perifériákról az adatokat a memóriába kell átmásolni, ezért a megbízható, gyors adat továbbítás kulcs fontosságú.

- Paralell ATA (PATA) / IDE

egy kábelre két eszköz master, slave
- Serial ATA (SATA)
- I/O vezérlő, DMA

3.4. Adattároló eszközök felosztása, fájlrendszerek

3.4.1. Partíciók

Lehetőségünk van arra, hogy a merevlemez lemezén lévő területeket felosszuk és különböző méretű, de egybe tartozó területeket egységbe foglaljunk. Egy ilyen területet partícióknak hívunk.

Minden partíció külön kezelhető a többitől, törölhető, formázható, másolható valamint saját fájlrendszerrel rendelkezik. Ezért fizikailag egy lemezen tárolhatunk különböző operációs rendszereket, anélkül, hogy zavarnák egymást.

A partíciók MS Windows alatt betűvel címkézve jelennek meg, pl. C, D. - fontos megjegyezni, hogy ha fizikailag másik lemezen van egy partíció az a meghajtó betű jeléből nem kideríthető. (pl. elképzelhető, hogy a C meghajtó egy partíció, amely a teljes merev merevlemezt elfoglalja, míg a D, illetve E meghajtók fizikailag egy lemezen helyezkednek, de valamilyen arányban megosztják a lemez területét). GNU/Linux alatt már tisztább a helyzet, a *dev* könyvtár tartalmazza a számítógéphez csatolt perifériák eszközfájljait, így a merevlemezekét is. A IDE csatolóval rendelkező lemezeket HDA, HDB, HDC, HDD névvel találjuk a könyvtárban, míg a SATA csatolóval renelezőeket SDA, SDB, SDC, stb. Általában két IDE csatlakozó van egy alaplapon, amelyre két-két eszközt lehet csatlakoztatni, ezért a HDA az elsőleges IDE csatlakozó master eszköze, a HDB ugyanezen kábelen lévő slave eszköz. A HDC, HDD a másodlagos IDE csatlakozóra felfűzött eszközöket jelzik. Ha a HDA lemez partíciókat tartalmaz, akkor az elsőleges partíció HDA0 néven fog szerepelni a *dev* könyvtárban, míg a második partíció HDA1 néven. SATA eszközök esetén az SD után következő A,B,C,D betűjelek és a 1,2,3,4 számok ugyanezt jelentik.

Természetesen az operáció rendszert informálni kell arról, hogy milyen partíciók léteznek az adott lemezen, ezeket az információkat tartalmazza a Master Boot Record (MBR). Ezen bejegyzés tartalmazza a partíciók méretét, kezdő és vég értékét valamint azt, hogy melyik partíció tartalmaz operációs rendszer elindításához szükséges adatokat - ezt/ezeket a partíciókat hívjuk bootolható partíciónak. A MBR-t az 1980-as években találták ki. Továbbfejlesztése a GUID Partion Table (GPT), amely számos kiterjesztést tartalmaz MBR-hez képest, például az MBR esetében a legnagyobb partíció mérete maximum 2.19 TByte lehet, míg az GPT esetén ez 9.4 ZetaByte (Zeta = 10^{21}). Tipikus háztartási méretekben ezek a számok megmosolytatónak tűnhetnek, de vegyük figyelembe, hogy ezeket a technológiákat nagyvállalai rendszerekben is használják, ahol egy adatbázis gond nélkül elérhet 2TB méretet.

3.4.2. Fájlrendszerek

A fájlrendszer feladata, hogy az eltárolandó fájlokat és könyvtárakat a winchester egy partícióján a megfelelő helyen elhelyezze, garantálja annak visszaolvashatóságát, valamint a változásokat adminisztrálja. Tehát a fájlrendszer funkciója kettős: egyrészt tárolja egy adott partíción lévő adatsávok (fájlok) helyét, másrészt kezeli az ezekhez kapcsolódó metaadatokat[4]. Minden, a fájlrendszerben tárolt adathoz (egy adott fájl fizikai elhelyezkedése a lemezen) tartozik egy metaadat bejegyzés is, ez tartalmazza a fájl vagy könyvtár nevét, létrehozás, módosítás dátumát, tulajdonos adatait, valamint a hozzáférési jogosultságokat. Fontos tudni, hogy a könyvtárak fizikailag nem jelennek meg a lemezen, azok csak a fájlrendszer adatbázisában lévő bejegyzésként vannak tárolva (tehát amennyiben egy lemezről elveszítjük a fájlrendszert leíró adatbázist, úgy a nyers adatok visszanyerhetőek, de: 1) nem fogjuk tudni, hogy melyik fájl hol kezdődött és hol van vége 2) nem fogjuk tudni rekonstruálni a könyvtárrendszert.) Látható, hogy a fájlrendszer helyes működése létfontosságú a tárolt adatok használhatóságának szempontjából, éppen ezért a fájlrendszer adatbázisa a lemezen általában több példányban, sokszorosítva kerül eltárolásra, csökkentve a megsérülés valószínűségét.

Mivel a számítógépeknek számos különböző feladatra használhatóak (családi személyi számítógép, bankszámlakezelő rendszer, egy tőzsdei kereskedőrendszer vagy egy milliós forgalmú webkiszolgáló), ezért az adatok tároláskor is különböző igények léphetnek fel (mind teljesítmény, mind biztonság tekintetében). Ezen igényekre kielégítésére rengeteg fájlrendszer megvalósítás létezik ebből ebben a jegyzetben részletesen a FAT, illetve ext2 fájlrendszerrel fogunk megismerkedni.

Fájlrendszereket általában az operációs rendszerrel együtt szokták szállítani, például MS Windows operációs rendszerhez két fájlrendszer érhető el: a FAT, illetve az NTFS. Míg Linux alatt a legelterjetebb az ext3 fájl rendszer, de számos egyedi igényéket kielégítő megvalósítás érhető el (xfs, jfs, Google file system, stb). Továbbá léteznek hálózati fájlrendszerek is, amelyek az operációs rendszer szempontjából átlagos partíciónak tűnnek, de fizikailag az adatok nem az adat számítógép merevlemezén tárolódnak. Például az egyetemen lévő tárhelyeinket (turdus, users)

[4] adatokat leíró adatok

is felcsatolhatjuk meghajtóként az általunk használt operációs rendszerben. Érdekesség, hogy linux alatt lehetőség van a Google által üzemeltetet gmail levelező szolgáltatáshoz tartozó postafiók tárhelyét meghajtóként felcsatolni, így arra bármilyen adatot menthetünk a szabad tárhely függvényében.

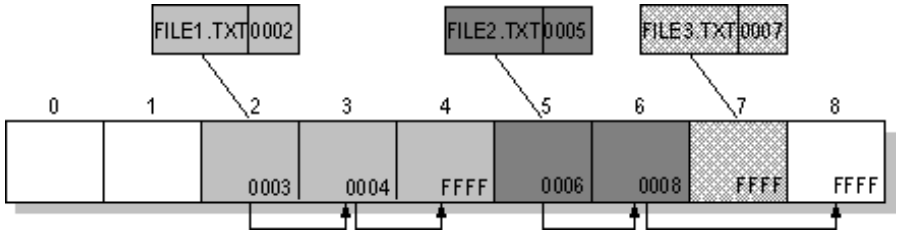
A fájlrendszerek egy fontos tulajdonsága a legkisebb foglalási egység, azaz ha létrehozunk egy üres fájlt, akkor fizikailag mekkora tárterület foglalódik le ennek a fájlnak a winchesteren.

Fontos megjegyezni, hogy a fájlrendszerek fájl- és könyvtárelhelyezési implementációja nem azonos az általunk megszokott könyvtárstruktúrával! Tehát a felhasználók számára látható könyvtárstruktúra mögött a fizikai tárolási módja ettől teljesen eltérő, ahogy ezt látni fogjuk.

3.4.3. Fájlrendszer implementációk

FAT fájlrendszer

A File Allocation Table (FAT) fájlrendszer a Microsoft fejlesztette és a Windows XP operációs rendszerig, régen ez volt a Windows operációs rendszerek által kizárólagosan használt fájlrendszer. A FAT fájlrendszer klasztereket tart számon, és a különböző FAT verzió főként abban különbözőnek, hogy hány biten tárolják a klaszterek sorszámait (FAT12, FAT16, FAT32). Ez bitszám határozza meg, hogy összesen mekkora lehet egy FAT partíció mérete. Az operációs rendszert is tartalmazó FAT partíciók tartalmaznak boot sectort is, ezt a szektor kerül beolvasásra memóriába az operációs rendszer bootolásának első lépéseként. Az alábbiakban részletesen megnézzük, hogyan tárolja a fájlokat a FAT fájlrendszer. A 3.2 ábrán látható a partíció egy darabja: a tárolt fájlok, valamint a hozzájuk tartozó allokációs tábla-beli bejegyzések. Az allokációs tábla - többek között - tartalmazza a fájl nevét és annak a klaszternek a számát, ahol a fájl kezdődik. Minden klaszter végén található egy cím amely a fájl többi darabját tároló klaszterre mutat vagy egy 0xFFFF jelzés, ami azt jelenti, hogy ez az utolsó klaszter, amiben a fájl részlete volt eltárolva. Fontos észrevennünk, hogy a rendszer nem követeli meg, hogy egy nagyobb méretű fájl egymás utáni klaszterek sorozataként legyen a lemezen: az operációs rendszer utasításaitól függően akár rengeteg, a lemez különböző pontjain elhelyezkedő klaszterbe is kerülhet a fájl egy-egy darabja. A FAT fájl rendszerekben a legnagyobb tárolható fájl 4GB.



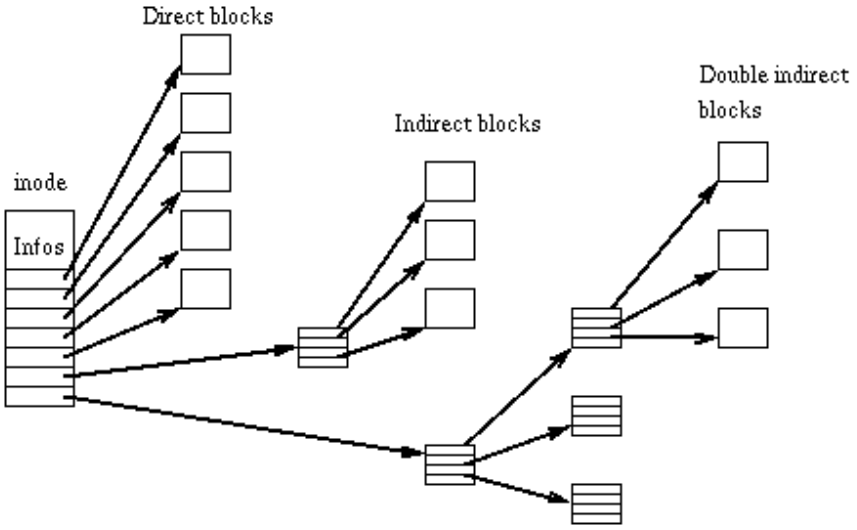
3.2. ábra. Egy FAT partíció darabja. Forrás: http://www.ntfs.com/images/recover-FAT-structure.gif

Ext2 fájlrendszer

Az ext2 fájlrendszer alapegysége a blokk, amelynek mérete tipikusan 1-8 Kb-ig terjed (ez a fájlrendszer létrehozásakor beállítható érték, de később nem változtatható és minden blokk ekkora méretű lesz). Így ha létrehozunk egy fájlt amiben elhelyezünk 2 karaktert az minimum 1 Kb-ot (vagy épp 8Kb-ot, ha akkora a blokkméret) fog elfoglalni a lemezen. Létezik egy szuperblokk, amely partíció elején helyezkedik el és többek között az operációs rendszer bootolásához szükséges információkat tartalmazza. A blokkokat csoportokban tárolják, ezzel is csökkentve a töredezettség mértékét.

A ext2 fájlrendszerben minden fájlt és könyvtárat egy úgynevezett inode ír le. Az inode tartalmazza a fájlal vagy könyvtárral kapcsolatos adminisztratív információkat: fájlnevét, létrehozás, módosítás dátumát, tulajdonost, jogosultságokat, stb. Az inode többi része 12-15 linket (blokk címeket) tartalmaz, amely egy csoportot címez meg, ezek a direkt blokkok (lásd a 3.3. ábrán). Amennyiben a fájl mérete meghaladja a direkt blokkokban tárolható adatmennyiséget,

akkor az utolsó link helyére nem egy direkt blokk címet helyezünk az inode-ban, hanem egy másik csoportleírót, ami további blokkokra vagy csoportleírókra mutat. Ezzel a módszerrel a legnagyobb tárolható fájl mérete 1KB-os blokk méretnél 16 GB, míg 8 KB blokkméret esetén 2 TB.



3.3. ábra. Az ext2 inode felépíése Forrás: http://upload.wikimedia.org/wikipedia/commons/a/a2/Ext2-inode.gif

A fájlok és könyvtárak mellett létezik egy másik típusú inode bejegyzés-típus is, ez a link. A link nem más, mint egy bejegyzés a fájlrendszerben, amely egy másik fájlrendszer-beli bejegyzésre hivatkozik. Önmagában tehát nem tárol adatot, hanem az őt megnyitó programot továbbirányítja az általa mutatott fájlra (ennek nagyságrendekkel butább utánzata az MS Windows-beli parancsikon). Két típusát különböztetjük meg, az egyik a szoft link a másik a hard link. A hard link esetében az inode-ban található linkek nem blokokkra mutatnak, hanem egy másik inode bejegyzésre. Ezzel szemben a szoft link egy speciális fájl, amely annak a fájlnak az elérési útját tartalmazza, amire mutat.

Ebből következően a hard linknél a mutatott fájl vagy könyvtár addig nem törölhető, amíg létezik rá mutató link (ezt a link számlálóból tudja - lásd feladatok).

A szoft linknél a link az elérési utat tárolja, így a mutatott fájl vagy könyvtár nem tudja, hogy létezik olyan hivatkozás, amely reá mutat. Éppen ezért ha azt letöröljük a hivatkozott fájlt, a link “célpont” nélkül marad, és “törött” link jön létre. Másik fontos különbség, hogy hard linket csak partíción belül lehet létrehozni, mivel az inode-ra mutat, aminek a számozása partíciónként újrakezdődik. Ezzel szemben a szoft link elérési utat tárol (ahogy azt már említettük egy partíció a könyvtárstruktúra tetszőleges pontjára becsatolható), így a szoft linket nyugodtan mutathat másik partíción elhelyezkedő fájlra.

SWAP fájlrendszer

Egy adott pillanatban nem minden programot használunk, amit elindítottunk a számítógépünkön, illetve az adott programnak sem használjuk minden részét. Ebből kifolyólag a nem aktív programokat, valamint programrészeket az operációs rendszer nem a viszonylag szűkös memóriában tartja, hanem a winchesteren, az úgynevezett swap (csere) területen. A tárolás olyan formátumban történik, hogy a kiírt programrészleteket ill. adatokat szükség esetén külön keresés-konvertálás nélkül a memóriába tudja visszatölteni. (Például: amikor a tálcára leteszünk egy programot és sokáig nem foglalkozunk vele, majd később elővesszük azt tapasztaljuk, hogy elég lassan reagál a kéréseink, de a winchester nagy tempóban dolgozik, ekkor kerül vissza a swap (csere) területről a memóriába az adott programhoz tartozó adatok és program részletek). MS Windows alatt a fájlrendszerben egy reguláris fájlként jelenik meg a swap terület, amit Pagefile-nak

hív a rendszer. Ez komoly hátrány, hiszen mint reguláris fájl, ez is ki van téve a töredezettségnek, hasonlóan a többi, fájlrendszer-beli fájlhoz. (Elérhetők olyan kis programok, amik a windows indulásakor töredezettségmentesiti, ezzel jelentős sebesség növekedés érhető el). GNU/Linux rendszereknél a swap tárterület egy linux-swap típusú fájlrendszerrel rendelkező külön partíció. Ez a megoldás természetesen nem szenved az előbb említett, a fájlrendszerek töredezettségéből fakadó hátránytól.

Cserélhető eszközök fájlrendszerei

Természetesen a cserélhető lemezeken is található fájlrendszer. CD és DVD esetében ISO9660 fájlrendszert szokás használni, még

Hálózati fájl rendszerek

- sshfs

Naplózó fájlrendszerek

Journaling, ext3, atomi műveletek fogalma

3.5. Könyvtárstruktúra és a fájlrendszer adminisztrációjának manipulációja

Linux alatt a BASH shell segítségével lehetőségünk van a parancssori utasítások segítségével fájlok és könyvtárak létrehozására, módosítására, valamint törlésére, tehát a könyvtárstruktúra módosítására. Továbbá lehetőségünk van a fájlrendszer adminszitrációs információk megjelenítéséra, megfelelő jogosultság esetén módosítására.

3.5.1. Könyvtárstruktúrával kapcsolatos parancsok

- könyvtárváltás: cd (change directory), aktuális pozíciónk a könyvtárfában: pwd (path to working directory), könyvtárfa megjelenítése: tree parancs. könyvtár tartalmának listázása: ls. Könyvtárakat az mkdir NÉV (make directory) parancs segítségével tudunk létrehozni.

- fájlok tartalmának megjelenítése, összefűzése: cat, szerkesztésre rengetek lehetőségünk van, például: mcedit, vim. Üres fájlokat is létrehozhatunk a touch parancs segítségével - létrejön az inode tábla, de a blokkok üresen maradnak.

- fájlokat másolni a copy, azaz cp MIT HOVA paranccsal lehet, áthelyezni a mv (move) paranccsal lehet. Könyvtárakat, és tartalmukat másolni a cp -r MIT HOVA utasítással tudunk (-r: rekurzívan, az alkönyvtárakat is másolja)

3.5.2. A fájlrendszerrel kapcsolatos parancsok

- Az egyes bejegyzések (fájlok és könyvtárak) hozzáférési jogosultságát a chmod paranccsal állíthatjuk be, a tulajdonos váltása chown parancsok segítségével történik, valamint felhasználó csoport váltása a chgrp parancs segítségével lehetséges.

- Az fsck parancs segítségével lehet ellenőrizni, hogy a winchester tartalma megegyezzik-e az adminsztrációs fájlok által leírt állapottal, azaz a fájlrendszer koherens állapotban van-e. Ilyen például akkor fordulhat elő, amikor hirtelen kikapcsol a számítógép (pl. áramszünet esetén) és valamilyen lemezművelet félbeszakad. Szintén problematikus eset a fájlrendszer koherenciájának szempontjából, ha akkor távolítunk el egy cserélhető eszközt, amikor még nem fejeződött be a lemezre írás művelet.

- Fájlrendszer egy üres partícióra a mkfs parancs segítségvel tudunk létrehozni, a parancs lefutása létrehozza az összes adminisztrációs állományt, ami szükséges a fájlrendszer menedzseléséhez. Hasonlóan, ha egy fájlrendszerrel rendelkező partíciót leformázunk, akkor a formázás létrehozza az üres adminisztrációs fájlokat (fontos, hogy ezzel még az előző fájlrendszerben tárolt adatok megmaradnak, csak nem tartozik hozzájuk adminisztrációs állomány).
- Az érvényes, hibamentes fájlrendszereket tartalmazó partíciókat használat előtt fel kell csatolnunk a könyvtárstruktúrába. Általánban az mnt könyvtárban van egy - partícióhoz tartozó - üres könyvtár, ahová a mount paranccsal tudjuk becsatolni a partíciót.

3.6. Feladatok

- nézzük meg mit csinál a df parancs, keressük meg a man oldalán, hogy mit csinál a -T kapcsoló és futtatssuk a df -T parancsot.
- nézzük meg a stat parancs man oldalát, próbáljuk ki a következőekre: sima fájl, könyvtár, eszközfájl, szotflink, hardlink.
- szintén nézzük meg az ls -i parancsot, keressünk egy fájl a könyvtárból és nézzük meg, hogy az ls -i parancsban megadott inode szám egyezik-e a stat parancs kimenetével.
- hozzunk létre szoft és hard linkeket, fájlra, könyvtárra, figyelünk a linkcounter értékének változására. Töröljük azt a fájlt amire a link mutat mit tapasztalunk szoft illetve hard link esetén?
- nézzük meg a dumpe2fs parancsot és futtassuk az egyik partícióra. A grep parancs segítségével (grep -i superblock) nézzük meg hány példányban tárolódik a lemezen a szuperblokk.

4. fejezet

Folyamatok és szálak

Bérci Norbert

4.1. Az operációs rendszer

Az operációs rendszerek célja, hogy a rendelkezésre álló erőforrásokat (helyi hardver erőforrásokat: processzort, memóriát, adattároló egységeket, megjelenítő egységeket, egyéb perifériákat; illetve a távolról, hálózaton keresztül elérhető erőforrásokat) elérhetővé tegye a felhasználói programok számára, gondoskodjon ezek hatékony és biztonságos felhasználásáról.

Modern operációs rendszerekben, modern hardveren (látszólag vagy ténylegesen) egyidejűleg több felhasználói program is futhat, ebbe beleértendők a több, különböző felhasználó által futtatott programok is, azaz több felhasználó is használhatja egy időben az erőforrásokat. Egy felhasználós [single user] illetve egyszerre csak egy programot futtatni képes [single task] operációs rendszerekkel nem foglalkozunk, kizárólag több felhasználót kezelő [multi-user] és több programot (kvázi-)párhuzamosan futtatni képes [multi tasking] operációs rendszereket tárgyalunk. Jelen jegyzetben a GNU/Linux operációs rendszerrel foglalkozunk részletesebben, de az általános megállapítások nagy része, a specifikusak egy része más operációs rendszerekre is igaz.

Az erőforrások felhasználásával kapcsolatosan ebben a jegyzetben a CPU ütemezést tárgyaljuk részletesebben, a memóriakezelési funkciókkal egy későbbi jegyzet foglalkozik. A tématerület az *Operációs rendszerek* tárgyban mélységében ismertetésre kerül.

4.1.1. A CPU ütemezés

Az operációs rendszer a processzor (számítási) erőforrásait felosztja, ütemezi [schedule] az éppen futó programok között. Ez általában úgy történik, hogy minden éppen futó program valamennyi ideig kizárólagos használatra megkapja a processzort, majd más programok futtatása következik. Ezt időosztásos [time sharing] működési módnak nevezik. Ha a programok számára kiosztott időszelet elegendően rövid, az ember számára úgy tűnhet, hogy a programok párhuzamosan futnak, akár még egy olyan (régebbi) gépen is, ami csak egyetlen processzort (és egyetlen processzor magot) tartalmazott, így a valódi párhuzamos futtatás fizikailag lehetetlen. Napjaink több processzoros (otthoni környezetben inkább csak több magos) hardverei esetében itt inkább arra kell gondolni, hogy több programot akarunk futtatni, mint ahányat a hardver fizikailag párhuzamosan futtatni képes. A továbbiakban a párhuzamos futtatást mindig ilyen értelemben értjük.

A párhuzamosan futó programok között a váltás történhet fix időnként is, de általánosabb, hogy akkor történik meg a váltás, ha a program már kellően sokáig futott vagy olyan erőforráshoz akar hozzáférni, ami jelenleg nem elérhető: például be akar olvasni egy fájlt valamilyen háttértárról. Ekkor a processzornak várakozni kellene (mivel a háttértárak általában sokkal lassabban képesek az adatokat beolvasni, mint amit a CPU képes lenne feldolgozni), így az operációs rendszer úgy dönt (a hatékonyabb CPU kihasználás érdekében), hogy egy másik, I/O műveletre nem váró program kapja meg a vezérlést. *Preemptív* [preemptive] multitaskingnak nevezzük azt az eljárást, amikor az operációs rendszer a program hozzájárulása nélkül tudja a program futását ideiglenesen megszakítani (majd egy későbbi időpontban folytatni).

4.1.2. RTOS

Valós idejű az operációs rendszer [real-time operating system - RTOS], ha garanciákat tud adni arra, hogy egy program valamely esemény bekövetkezte után mennyi idővel kezdi meg a futását, illetve a programok képesek processzor időt előre lefoglalni, azaz adott időnként biztosan hozzáférni az erőforrásokhoz. Az előbbi tulajdonság számítógép vezérelt rendszerek esetében fontos, hiszen például egy ipari rendszerben az egyes fizikai paraméterek megváltozására rövid időn belül reagálni kell, ellenkező esetben akár visszafordíthatatlan folyamatok is lejátszódhatnak (például egy túlnyomás érzékelő jelére későn reagáló vezérlő rendszer már nem tudja időben kinyitni az elvezető szelepeket és robbanás következik be). Az utóbbira példa egy audiovizuális tartalom megjelenítő program (például film lejátszó), aminek az egyes képkockák megjelenítéséhez rendszeresen, adott időközönként hozzá kell férnie a processzorhoz (és a megjelenítő eszközhöz, adattárolókhoz, stb.) hogy folyamatos legyen a lejátszás. Ezek a problémák nyilván csak akkor jelentkeznek, ha az operációs rendszer több programot is futtat egyszerre, és azok az erőforrásokat jelentős mértékben használják is: ha egyetlen program fut, senki sem veheti el a processzort a programtól. Az operációs rendszer *soft real-time*, ha az előző célokat próbálja elérni, és *hard real-time*, ha erre garanciát is vállal.

4.1.3. Rétegezett felépítés

Nagyon fontos következménye az időosztásos működésnek (is), hogy a programváltások miatt a programok nem használhatják a hardvert közvetlenül, kizárólag az operációs rendszeren keresztül. A legfelső réteg tehát az alkalmazói programok, alatta foglal helyet az operációs rendszer, és legalul a hardver.

A különböző perifériák, hardver elemek működését ún. eszközkezelők [device driver] biztosítják, amik az operációs rendszerbe beépülve lehetővé teszik, hogy azokat a felhasználói programok is elérhessék.

A programok az operációs rendszer funkcióit programozási interfészeken [application programming interface - API] keresztül érik el. Ez Linux, UNIX esetében SO (shared object), Windows esetben DLL-eken (dynamically linked library) keresztül valósul meg. Linuxban az `ldd` paranccsal megjeleníthető, hogy a parancssorban megadott bináris állományok melyik SO-kkal linkelődnek össze, ha elindítjuk őket:

4.1.1. példa. `$ ldd /bin/bash`

```
linux-vdso.so.1 =>  (0x00007fff3b5fe000)
libncurses.so.5 => /lib/libncurses.so.5 (0x00007f4a505f7000)
libdl.so.2 => /lib/libdl.so.2 (0x00007f4a503f3000)
libc.so.6 => /lib/libc.so.6 (0x00007f4a50091000)
/lib64/ld-linux-x86-64.so.2 (0x00007f4a50832000)
$
```

4.2. Folyamatok és szálak

*Folyamat*nak [process] nevezzük valamely végrehajtható állomány operációs rendszer által futtatott példányát. Modern operációs rendszerekben minden folyamat saját memóriaterülettel rendelkezik, amit szabadon felhasználhat, de másik folyamat adatát alapértelmezetten nem érheti el.

4.2.1. feladat. Milyen hibát jelent Windows XP esetében a "XXXX has encountered a problem and needs to close. We are sorry for the inconvenience." üzenet?

Az aktuálisan futó folyamatokról a `ps` paranccsal tudunk információt kérni:

4.2.1. példa.

```
$ ps
  PID TTY          TIME CMD
  774 pts/0    00:00:00 ps
32752 pts/0    00:00:00 bash
```

```
$
```

Ha a saját folyamataink kívül más folyamatokat is ki akarunk listázni, a `ps ax` parancsot használhatjuk:

4.2.2. példa.

```
$ ps ax
  PID TTY      STAT   TIME COMMAND
 5887 tty4     Ss+    0:00 /sbin/getty 38400 tty4
 5888 tty5     Ss+    0:00 /sbin/getty 38400 tty5
 5890 tty2     Ss+    0:00 /sbin/getty 38400 tty2
 5892 tty3     Ss+    0:00 /sbin/getty 38400 tty3
 5894 tty6     Ss+    0:00 /sbin/getty 38400 tty6
 5972 ?        Ss     0:00 /sbin/syslogd -u syslog
 6023 ?        S      0:00 /bin/dd bs 1 if /proc/kmsg of /var/run/klogd/kmsg
 6025 ?        Ss     0:00 /sbin/klogd -P /var/run/klogd/kmsg
 6046 ?        Ss     0:00 /usr/bin/dbus-daemon --system
 6060 ?        Ss     0:00 /usr/bin/system-tools-backends
 6118 ?        Ss     0:00 /usr/sbin/sshd
 6149 ?        Ss     0:00 avahi-daemon: running [kanape.local]
 6150 ?        Ss     0:00 avahi-daemon: chroot helper
 6195 ?        Ss     0:00 /usr/sbin/cupsd
 6230 ?        S      0:00 /bin/sh /usr/bin/mysqld_safe
 6273 ?        S      0:00 logger -p daemon.err -t mysqld_safe -i -t mysqld
 6367 ?        S      0:06 postgres: writer process
 6368 ?        S      0:00 postgres: stats buffer process
 6369 ?        S      0:01 postgres: stats collector process
 6405 ?        Ss     0:08 postgres: writer process
 6406 ?        Ss     0:01 postgres: stats collector process
 6463 ?        S      0:00 /usr/sbin/inetutils-inetd
$
```

4.2.2. feladat. A `man ps` parancs segítségével keressük meg a dokumentációban, hogy mit jelent a STAT oszlopban az S, R, Z státusz!

4.2.3. feladat. A `ps axl` parancs kimenetében mit jelentenek a UID, PID, PPID oszlopok?

4.2.4. feladat. Mire szolgál a `top` parancs?

4.2.5. feladat. Hogyan lehet a `nice` és a `renice` parancsokkal egy folyamat nice level -ét megváltoztatni? (Ismét használjuk a `man` parancsot a dokumentáció elolvasásához!)

5. fejezet

Memóriakezelés és architektúrák

Tuza Zoltán

5.1. Bevezetés

A fejezet célja, hogy programozói szempontból bemutassa az elsődlegesen használt illékony memóriát, azok szervezését, felépítését. A második részben az Intel cég x86-os architektúráján keresztül mutatjuk be azokat az elveket és megoldásokat amik a mai processzorokat jellemzik. Végül a fejezet zárásaként tárgyaljuk a buszrendszereket, amin keresztül a CPU és a memória egymással kommunikál. A fejezetben előkerülő témák mindegyikéről könyvtárakat megtöltő irodalmat lehet találni, ezért sok esetben egyszerűsítünk, elhanyagolunk részeket. Ezek a részek más tantárgyakban hangsúlyosan fognak előkerülni, a mostani célunk egy általános szemlélet megalapozása.

5.2. Memória

Memóriák esetében két nagy csoportot különböztetünk meg: az egyik az úgynevezett illékony memória, mely a tápfeszültség megszűnésével elveszti a benne tárolt információt, ilyenek például a számítógépek elsődleges memóriái, pl DRAM, SRAM stb. A másik az csoportot a nem-illékony memóriák adják. Ezek a pendrive-ok, feldrive-ok, valamint az alaplapok BIOS programját tároló memóriák. Amint azt már az adattárolás fejeztnél is említettük, a nem-illékony memóriák írási és olvasási sebessége különbözők[a], ez az ára a tápfeszültség-mentes tárolásnak. Viszont egy nagy teljesítményű központi számító egységet (CPU) folyamatosan adatokkal kell ellátni, ezért használunk gyorsan írható és olvasható illékony memóriákat erre a célra (pl. a ma használatos DDR3 memóriák 8Gigabit/sec adatátvitelre képesek). Továbbiakban csak az illékony memóriákkal foglalkozunk. A memória alapegysége a cella, mely egy bit (logikai 0 vagy 1) tárolására alkalmas (gyakorlatilag egy tervezérlésű tranzisztor állapota határozza meg a tárolt bit információját - részletesen Áramkörök elmélet és számítása című tárgyban). A cellákat fix szélességű szavakba szervezik, ebből következően lehetnek 1, 2, ..., 32, 64, ... bit széles szavakból álló memóriák. Ezeket a szavakat N bit hosszú, linárisan nővekvő címmel tudjuk megcímezni. Ebből következően egy memória mérete (tárolható bitek száma) a következően határozható meg: $2^N * (\text{szószélesség})$. Ezek alapján meghatározhatjuk, hogy a processzor és memória között milyen fajtájú és számú összeköltetésre van szükségünk. Először is szükséges N darab összeköttetés, amelyek gyűjtő névvel *címbusz*nak hívunk, ez egy egy irányú összeköttetés a processzor irányából a memória irányába. Továbbá szükségünk vagy egy kétirányú összeköttetésre, amely egy szó szélességű (azaz annyi párhuzamos elemből áll, amilyen széles szavakba van szervezve az adott memória (pl 32 bit, 64 bit stb.), így egy ütemben egy teljes szó közlekedik a memória és a processzor között). Ezt nevezzük *adatbusznak*. Ezen kívül az adatátvitel engedélyezéshez és időzítéséhez további összeköttetések szükségesek, melyeket összefoglalóan kontrollbusznak hívnak. A buszokról a fejezet végén részletesen beszélünk.

Kérdés: hány bájt információ tárolható egy 8bites szószervezésű és 6 bittel címezhető memóriában?

Fontos megjegyezni, hogy a memória méretének növekedésével a szükséges cím- és adatvezetékek száma jelentősen megnőhet, ezért más megoldásokat használnak a mai számítógép gyártók. Erről részletesen a Digitális rendszerek és Architektúrák tárgyban lesz szó.

Amennyiben rendelkezünk egy adott N címhosszúságú és w szóhosszúságú memóriával, úgy a következő fontos kérdés, hogy hogyan tudunk olyan adatokat ábrázolni, melyek hossza meghaladja a szóhosszúságot. Legyen a memória az egyszerűség kedvéért byte-szervezésű (8 bites szóhossz), tároljunk ebben a memóriában egy 32 bites számot. Az egyik lehetőség, hogy az első byte szélességű memória címre az tárolandó szám legmagasabb helyiértékű bitjeit tesszük (helyiérték szerint csökkenő sorrendben), ezt hívjuk *big-endian*-nak. A fordított sorrendet - amikor a tárolandó szám legkissebb helyiértékeit tartalmazó byte-ot tároljuk az első memória címen - nevezzük *little-endian*-nak.

Egy másik fontos fogalom az igazítás (alignment) fogalma, mivel a CPU és a memória között szó-szélességű a kommunikáció, ezért a több szón ábrázolt adatokat több részletben kell átküldeni az adatbuszon (lásd buszok szakasz). Mivel ez a kommunikáció arra van kihegyezve, hogy szavak egész számú többszörösét küldjük át, ezért hasznos, ha memóriában az adatok egymás után azonos számú szavanként következzenek. Adataink azonban nem mindig ugyan annyi bitből állnak (tehát nem ugyanannyi szót foglalnak el), ezért fordítási időben "meghizlaljuk" őket, hogy minden adat egyforma hosszú legyen. Például ha minden adat egyformán 16 szóból áll akkor memóriaolvasásnál 16-osával lehet léptetni a memóriacímeket (hiszen csak annyinként kezdődik új adat), ezzel javul a memória olvasás sebessége és a rendszer teljesítménye.

5.2.1. DRAM és SRAM

Az alábbiakban összehasonlítunk két félvezető technikával működő illékony memória megvalósítást.

A statikus ram cella (SRAM) esetében egyetlen bit tárolásához 6 tranzisztor kell, tápfeszültség folytonos jelenléte esetén a tárolt bit nem változik meg. Ezzel szemben a dinamikus ram cella (DRAM) egyetlen tranzisztorból és egy kondenzátorból áll, ezen kondenzátor állapota határozza meg a cella által tárolt bit értékét. Ezen típus megvalósítás hátránya, hogy a szivárgó áramok miatt a kondenzátor töltöttsége idővel csökken, és bizonyos idő után egy olyan szint alá esik, amikor már nem lehet biztonsággal megállapítani a benne tárolt információ logikai értékét. A frissítés idejére az adott cella nem érhető el. A dinamikus ram cella tehát - a statikussal szemben - a benne tárolt érték rendszeres frissítését igényli, továbbá mindkét cellatípus a csak tápfeszültség jelenlétében képes információt tárolni. Ezekből fakadóan a DRAM esetén különböző memóriaszervezési feladatok kell megoldani: célunk, hogy a memória egy részét frissíteni tudjuk, míg a másik része addig elérhető írás-olvasás céljára. Egy fontos fogalmat lehet ezen a ponton bevezetni, ez pedig a capacity-per-are, azaz az egységnyi területen elérhető tároló kapacitás fogalma. A DRAM esetén ez az érték megközeltőleg háromszorosa a SRAM-nak, többek között ennek is köszönhető, hogy a DRAM jelentősen elterjed a SRAM rovására, függetlenül a bonyolult memóriafrissítési igényektől.

5.3. A Central Processing Unit (CPU) bemutatása az Intel x86-os architektúrán keresztül.

A CPU alapvető feladata aritmetikai, logikai és adatmozgatási műveletek hatékony megoldása. Ehhez szükséges egy utasításkészlet, amivel utasíthatjuk a processzort egy olyan művelet elvégzésére, amit a processzor felépítése és kialakítása megenged. Ilyen lehet például két szám összeadása vagy kivonása, logikai "AND" számítása két számon, adatok áthelyezése a memória egyik rekeszéből egy másikba, adatok mozgatása a memória és a processzor között, vagy program futását befolyásoló utasítások: elágazások, feltétel nélküli ugrások (egy modern processzor utasításkészlete rengeteg - több ezer - utasításból áll). Ezeket az utasításokat bináris formában tároljuk és a processzorban található vezérlő dekódolja őket, majd utasítja a processzor megfelelő részegységét a feladat elvégzésre. Az utasítások által kijelölt műveletek az utasítás után felsorolt operandusokon hajtódik végre. Az egyes utasítások lehetnek 0, 1, 2, 3 operandussal rendelkező műveletek.

Például az `ADD AX, BX` utasítja a processzor aritmetikai és logikai egységét (ALU), hogy az AX és BX regiszter tartalmát adja össze. (Implementáció függő, de általában az eredmény az AX regiszterben keletkezik)

A processzorokban minden (rész)utasítás végrehajtása ütemre történik. Vannak olyan műveletek, amik egy ütem alatt is végrehajthatóak, de vannak olyanok, melyek több ütemet igényelnek. Ezt az ütemet a processzor órajele határozza meg. Az órajelet általában Hertz-ben mérjük, amely azt mondja meg, hogy egy másodperc alatt hát ütem zajlik le.

A főbb rész egységek a processzoron belül:

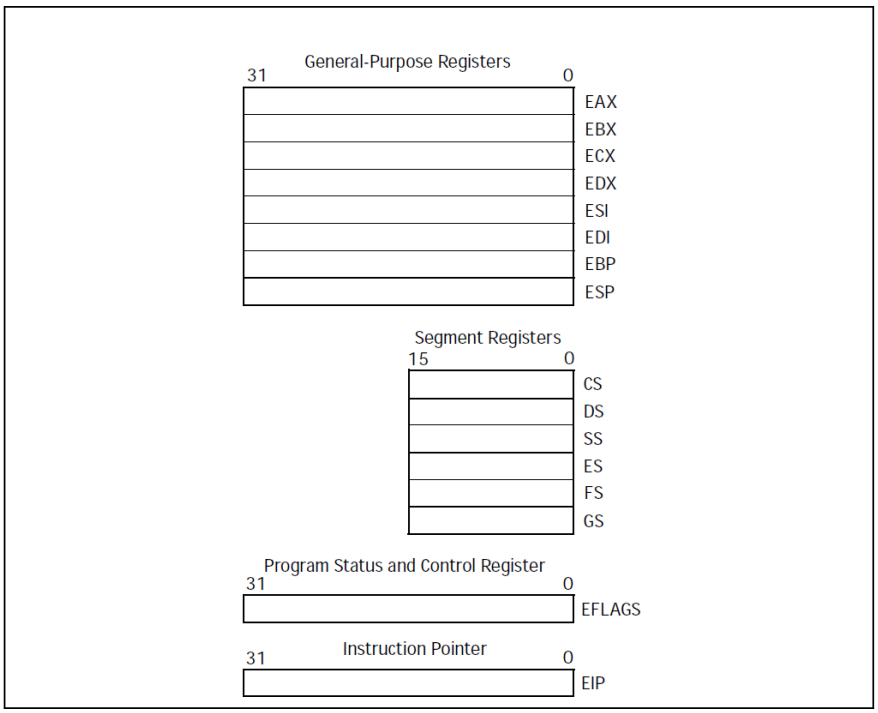
- Általános és speciális regiszter tömbök. A regiszterek nem mások mint rendkívül gyors memória rekeszek közvetlenül a processzorba építve. A regiszterek processzortípustól függően 8, 16, 32, 64 bit szószerverezésűek. A regisztereknek nevük van, amelyre programozás során hivatkozhatunk, sőt bizonyos regiszterek kisebb részeihez is hozzá férhetünk, például egy 32 bites regisztert használhatunk két darab 16 bites ként is. A 5.1 ábra mutatja be az általános és speciális célú regisztereket, céljukat lásd programok végrehajtása résznél. Az egyes logikai és aritmetikai műveletek elsődleges adat forrásai ezen regiszterek. Ha tehát két számot össze szeretnénk adni (ADD művelet), akkor azokat előtte el kell helyeznünk egy-egy regiszterben. Ezért is van szükség memória kezelő utasításokra, amelyet a memória kezelő egység hajt végre. (Van lehetőség közvetlen memóriacímekkel dolgozni, de abban az esetben is a fenti művelet sor hajtódik végre.)

- Memóia kezelő egység feladata, hogy kapcsolatot tartson a elsődleges rendszermemóriával (természetesen árnyaltabb a kép, mivel a mai processzorok többszintű gyorsítótárral (cache) rendelkeznek, de ezeket most didaktikai okokból elhanyagoljuk.) Az egyik legfontosabb memória kezelő utasítás a `MOV cél, forrás`, amely a forráson található adatot (ez lehet egy memória cím, vagy regiszter) a cél helyre másolja (szintén lehet regiszter vagy memória cím) Erre alább látható egy példa program:
  ```
  MOV AX,0006h
  MOV BX,AX
  MOV AX,4C00h
  ```
  A fenti utasítás sorozat betölti a `0006h` memória cím tartalmát az `AX` regiszterbe, majd ez az adatot az `AX` regiszterből a `BX` regiszterbe másolja végül, az `AX` tartalmát a `4C00h` memória címre írja.

- Utasítás dekódoló egység és vezérlő egység Az eddig bemutatott utasítások (ADD, MOVE, és még több száz másik) képzik a processzor utasításkészletét. Amikor egy adott programnyelven megírt forráskódot lefordítunk, akkor fordító az adott processzor utasításkészletét helyettesíti be a programozási nyelv utasításai helyére. Ezen utasítások halmaza úgynevezett gépi kód, amit a processzor közvetlenül megért. Ezen bináris sztringek értelmezését végzi az utasítás dekódoló egység. Az eddig mutatott utasítások (ADD, MOVE) ugynevezett mnemonikok, melyek egy-egy utasításkészletbeli elem bináris kódjához rendel értelmes szó vagy rövidítés. Minthogy (a Neumann-elv értelmében) a program is adatként van tárolva a memóriában, így első lépésként a memória kezelő egység által beolvasott utasítás a memóriából az utasításregiszterbe kerül. A dekódoló ezt kiolvasva állapítja meg az elvégzendő művelet. A vezérlő egység utasítja a megfelelő részegységet a kijelölt művelet elvégzésére. Egy ilyen lehet például két szám összeadása, vagy logikai művelet elvégzése.

- Aritmetikai és Logikai egység feladata a különböző aritmetikai és logikai műveletek lebonyolítása és ezen eredmények visszaírása a regiszterekbe. Szintén az ALU feladata, hogy minden elvégzett művelet után frissítse a státuszregiszterek tartalmát. Ez a regiszterben a műveletvégzés során esetlegesen bekövetkezett különleges eseményt hivatott jelezni (túlvagy alulcsordulás, volt-e carry az elvégzett művelet után stb.)

Most, hogy tisztázásra kerültek a főbb processzor-részegységek nézzük meg, hogyan történik egy program végre hajtása a processzorban.



5.1. ábra. Általános és speciális célú regiszterek az Intel x86 architektúrákon Forrás: IA-32 Architectures Software Developer Manuals

5.3.1. Utasítás végrehajtás

Első körben arra fogunk koncentrálni, hogy hogyan áramlik az információ a processzorba és azon belül. (Magának a program indulásának kérdését, nem tárgyaljuk, mert akkor részletesen kéne beszülnünk a hívási vermekről és az alprogramokról, melyekhez más tárgyakban megszerzendő ismeretekre kéne építeni). Második lépésben tárgyaljuk, hogy az x86 architektúra hogyan kezeli a memóriát.

Az első feltételezésünk a memóriáról, hogy a programunk számára két tartományra van osztva, az első részben az elvégzendő utasítások gépikódja található, míg a második részben azon adatok, amiken az utasítások végrehajtatjuk. A következő végrehajtandó utasítás memória címét az úgynevezett program számláló (PC) regiszterben tároljuk. Magát a végre hajtandó utasítást beolvasás után, utasítás regiszterben tároljuk (IR). A memória cím regiszter (MAR) tartalmazza azt a memória címet ahonnan a következő időpontban olvasást vagy ahová írást kell végezni. A beolvasott vagy a kiírandó adat a memória adat regiszterbe (MDR) kerül, vagy onnan íródik ki. A fentiek alapján egy utasítás beolvasása a processzorba a memóriából, a következő képen írható le (A leíráshoz a regiszter transzfer jelölést használjuk (RTN)):

$$\begin{aligned} MAR &\leftarrow [PC] \\ MDR &\leftarrow [Memory]_{MARaddress} \\ PC &\leftarrow PC + 1 \\ IR &\leftarrow [MDR] \end{aligned}$$

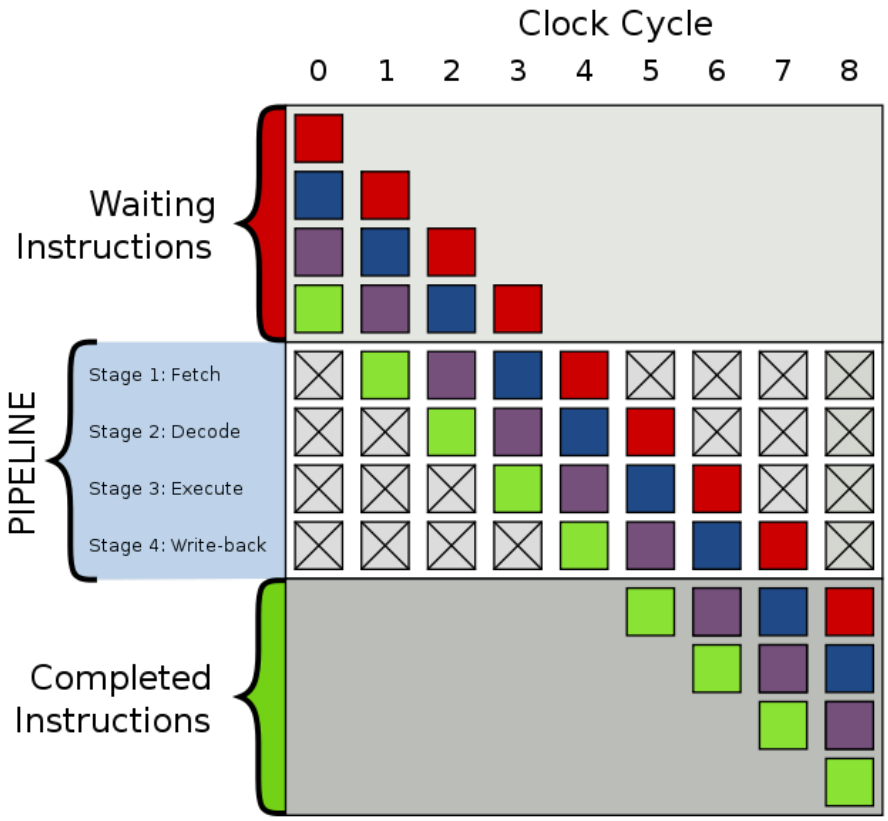
A fenti kódrészlet ábrázolja, hogy hogyan történik egy utasítás beolvasása. Látható, hogy a program számláló által mutatott memória címről, az utasítás az utasítás regiszterbe (IR) kerül. Amennyiben adatot olvasunk vagy írunk a memóriába, akkor is fenti művelethez hasonlóan járunk el, de ekkor a cél regiszter különböző lehet, pl AX, BX, SS, stb. Erre a utasítás végrehajtás során kerülhet sor.

A fenti művelet sort, lehívásnak/behívásnak (fetch) nevezzük, ezzel hívjuk be a memóriából a végrehajtandó utasítást. A következő lépés az IR-ben elhejezett utasítás értelmezése (decode), amely alapján a vezérlő utasítja a megfelelő végrehajtó egységet a végrehajtásra (execution). A

végrehajtás során lehetséges, hogy szükséges adatok beolvasása a memóriából, de az is lehet, hogy már a regiszterekben vannak az előző utasítás eredményeképpen. Végül a keletkező eredményt vissza kell írni a regiszterekbe. Ennek alapján egy művelet végre hajtását öt lépésre tudjuk szét bontani

- Fetch - utasítás beolvasása a PC által mutatot memória címről
- Decode - végrehajtandó művelet értelmezése
- Execute - utasítás végrehajtás az utasításnak megfelelő részegségen,
- Write-back - az utasítás eredményét visszaírjuk a megfelelő regiszterbe.

Amennyiben egy utasítás végrehajtását, szét tudjuk bontani ilyen önálló részfeladatokra, akkor lehetőségünk van párhuzamos végrehajtást megszervezésére, amit csővezetéknek (pipeline) nevezünk. Erre egy példa az autógyár gyártósora, minden munkaállomáson egy részt feladatot hajtanak végre, ami mindig azonos, például meghúznak egy csavart. A futószallag bizonyos előmeghatározott időnként tovább halad. A gyártósor végén pedig előáll a kész üzem kész autó. Pontosan ez történik a processzorban a fent felsorolt 4 lépést (Fetch,Decode,Execute,WriteBack) különböző rész egységeken végezzük el, melynek eredményeként egy utasítás végrehajtódik a processzoron. Ezt a folyamaton a 5.2 ábrán tudjuk követni.



5.2. ábra. Egy egyszerű pipeline felépítése, a különböző színek a különböző utasítás feldolgozó funckiókat jelölik Forrás: wikipedia.org

Ellentétben az autógyártósorral, a processzor képes egyes részutasításokat késleltetni, míg másokból többet megcsinálni. (Ez visszavezethető arra, hogy az egyes részutasítások végrehajtás különböző újra jel ciklus számot igényelnek) Ebből következően, hogy egy pipeline-t megfelelően működtethessük az utasítások részfeladatainak sorrendjét megfelelően kell szervezni. Ami azt eredményezi, hogy egyes utasítások nem abban sorrendben kerülnek feldolgozásra, ahogy azt

a programunkban megírtuk. Ettől kívülről nem változik meg a programunk viselkedése, de a proceszor kihasználtsága jelentősen javul azáltal, hogy az egyes részegységek, mindig foglalkoztatva vannak, nincs várakozás adatokra, erőforrásokra. Ezt a fogalmat nevezzük *out-of-order exectuion*-nak[1]

A pipeline mellett egy másik gyorsítási lehetőség az úgynevezett super scalar, ennél a módszernél az egyes funkciókat megvalósító állomásokból többet is elhelyezünk egy processzorban. Az autógyártós példánál maradva egyszerre több gyártó sort üzemeltetünk. Fontos megjegyezni, hogy ez egy processzoron belül történik és a processzor probálja úgyszervezni a végrehajtását egy programnak, hogy párhuzamosan végrehajtó elemek legyenek benne.

5.3.2. Memória modellek az x86-os architektúrán

A következő lépésben vizsgáljuk meg, hogy hogyan milyen memória kezelési módszerek vannak, amelyhez a 5.3 ábra fog a segítségünkre lenni. Az első módszer az úgynevezett lineáris címzésű memória modell. Ebben az esetben az egyes memória hozzáférésekhez használt cím közvetlenül kikerül a címbuszra (lásd busz rendszerek részt), és megcímzi a memória megfelelő rekeszét. A második módszer a memória szegmentálása különböző memória szelektekre (nem feltétlenül egyforma méretű szegmensekre). Ekkor egy memória rekesz megcímzéséhez két dologra van szükségünk, az első a szegmens választó (ez egy memória cím, amely a szegmenst tároló memória darabra mutat), a második egy eltolás, amely segítségével a szegmensen belüli eltolás adjuk meg (pl 55. rekesz a szegmensben). Ekkor egy cím fordítás (szegmens szelektor+eltolás) után alakul ki a fizikai cím és azt kerül ki az adatbuszra. A szegmentálás előnye, hogy különböző funkciójú szegmenseket tudunk kialakítani és ezt a processzor regiszterei is támogatják a szegmens regiszterek segítségével, lásd XX ábra. A szegmenseknek köszönhetően a programok azt hiszik csak ők vannak a memóriában, mivel nem látnak át másik szegmensekre, ezért nem is tudnak a programok egymásról. A három fő szegmens a kód szegmens, adat szegmens és verem szegmens. A kód segmensben található annak a szegmensnek a címe, amiről az utasításokat olvassuk és ebben az esetben az eltolás nem más mint az utasítás számláló értéke. Az adat szegmensekből olvassuk az aktuális programunk adatait. A verem szegmenst nem tárgyaljuk részletesen, mivel ehhez az alprogramok, függvény hívások ismeretét kellene feltételeznünk.

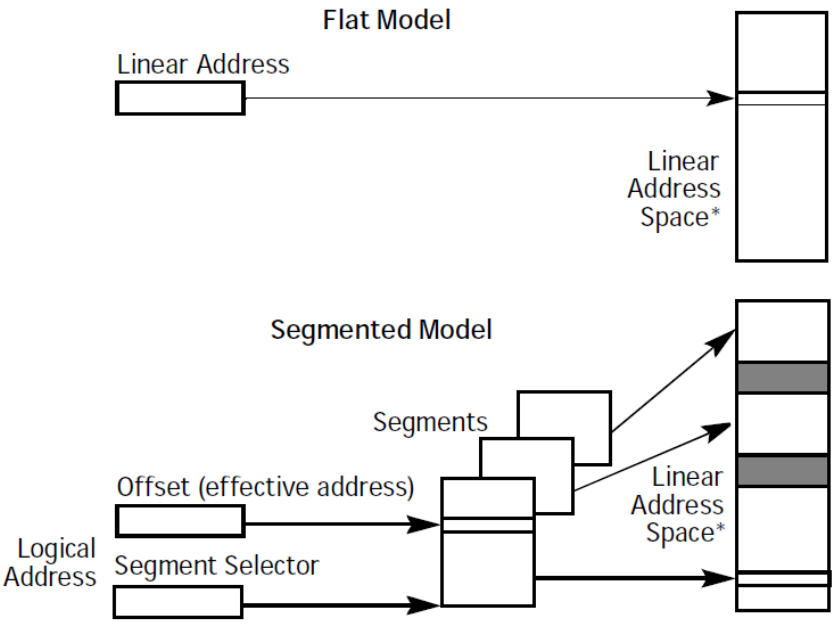
Megjegyzés: programozás során sokszor fogunk segmentation fault vagy access violation hibákat kapni, ezek főként akkor fordulnak elő, amikor olyan programozási hibából (pl tömb túl indexelés) egy olyan memória területre szeretnénk írni amihez a programunknak nincs joga. Tehát szegmens határt sért a programunk.

A memória hozzáférés modellnek van még egy szintje az lapozás, amit a virutális memória résznél fogunk részletesen tárgyalni. Fontos megjegyezni, hogy a szegmentáció a lapozás elterjedésével jelentőségét vesztette, ezért x86-64 64 bites architektúrák már nem használják a szegmentációt, csak a lapozást. (Kompatibilitási okokból, természetesen a szegmens regiszterek rendelkezésre állnak, de a 64 bites programok nem használják ezeket)

5.3.3. Virutális memória

A virutális memória egy másik céllal jött létre, de nagyon hasonló a megvalósítása a szegmentációhoz. A virtuális memória szintén processzor szinten van támogatva és a processzor memória kezelő egysége vége a szükséges adminisztrációt. A virtuális memória célja, hogy elsődleges memóriát kiterjessze a másodlagos tárolóra. Azaz a fizikai memória és a HDD együttesen egy nagy memóriának látszik, egy hatalmas cím tartománnyal. Ezt a cím tartományt lapokra osztjuk fel (tipikusan 4Kbyte egy lap mérete), és a programok és adatok ezeken a lapokon vannak elhelyezve. Mivel a cím tartomány egyik fele a fizikai memóriában a másik fele a merevlemezen található, ezért az optimális teljesítmény érdekében az aktuális programhoz tartozó lapokat fizikai memóriában célszerű tartani. Ez teremti meg a lapkezelési stratégia szükségességét, amely eldönt, hogy egy lap a fizikai memóriában vagy a merevlemezen legyen (azaz áthelyezze a virtuális memória cím tartományán belül máshova). Az, hogy egy lap a merevlemezen vagy a fizikai memóriában van azt cím fordítás közben tudja meg a processzor memóriakezelő egysége. Nézzük meg a 5.5 ábra segítségével, hogy hogyan fordítunk le egy virtuális memóriabeli címet fizikai címre. A

[1] Ne próbáljuk szószerint lefordítani, nem használaton kívüli kivégzésről van szó ;)



5.3. ábra. Memória kezelési modellek az Intel x86 architektúrákon. Forrás: IA-32 Architectures Software Developer Manuals

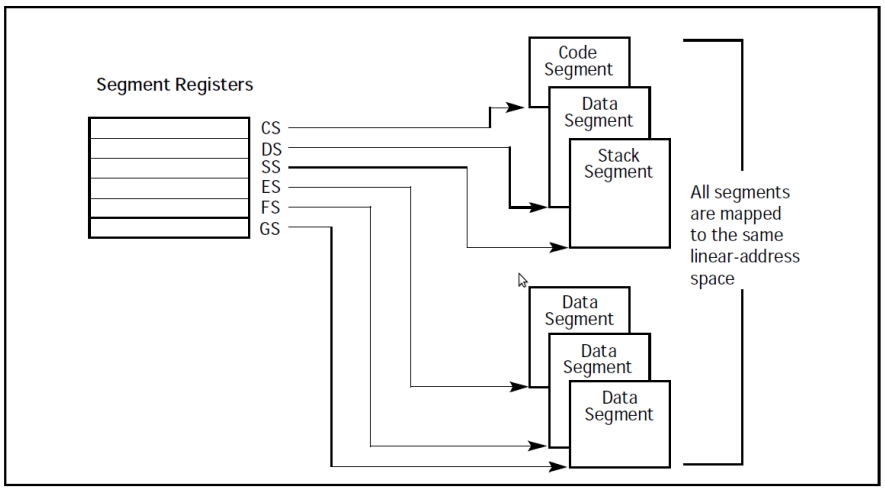
virtuális cím két részből áll egy könyvtár címből, amiben az összes lap kezdő címe megtalálható és egy eltolásból, hogy a lapon melyik bejegyzésre van szükségünk, ez bejegyzés tartalmazza a fizikai memória beli címét az kért adatnak. Amennyiben egy lapnak nincs fizikai memória beli címe, akkor page fault exception váltódik ki a virtuális memória kezelőben és elindul a lap behozása a fizikai memóriába, majd a tábla adatai frissülnek, többek között a fizikai memória beli címével. A lap könyvtár beli bejegyzés a lap kezdő címeken felül egyéb információkat is tartalmaz a lapról, az egyik ilyen az, hogy az adott lap futattható-e. Tehát megtudjuk mondani, hogy az adott lapon adat van és ezért a processzor ne próbálja meg azt utasításként értelmezni.

Ezt a szakaszt a Neumann és a Harvard architektúra összehaszonlításával zárjuk. A Neumann architektúrát Neumann János magyar származású matematikus javasolta 1945-ben. Ennek lényege, hogy az adatok és utasítások ugyan azonos memóriában léteznek bináris formában. Ami sok szempontból előnyös hiszen a programok tekinthetik a másik programot adatnak és azokat átírhatják. Egy másik megvalósítási lehetőség a Harvard architektúra, ahol külön adat és program memórai van, ennek előnye, hogy a két memória egymástól függetlenül hozzáférhető. Tehát még az egyik memóriából adatokat olvasunk az aktuális utasítás végrehajtására, addig a másik memóriából már a következő utasítást olvashatjuk be a processzorba.

SIMD VIMD MIMD

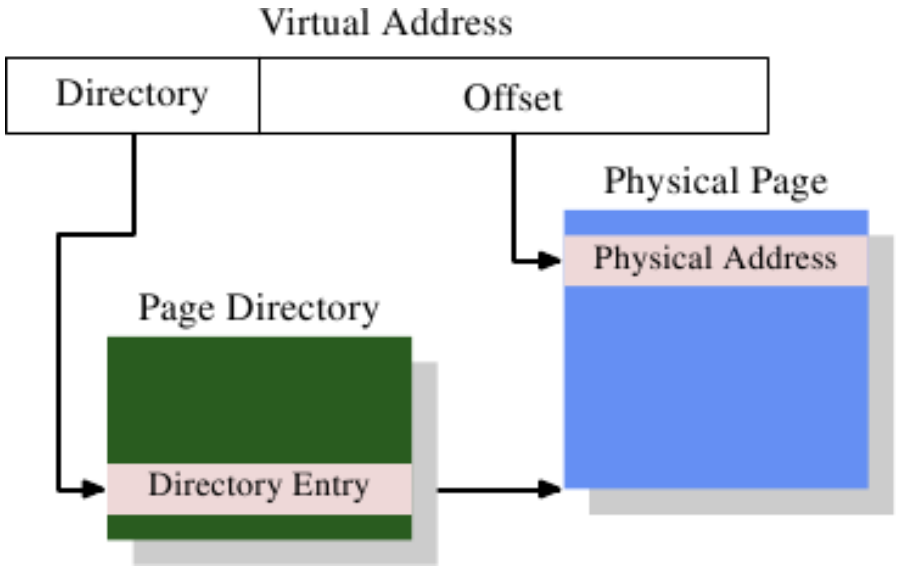
5.4. Adat- és Címbusz

Végezetül nézzük meg doboz szinten ebben a fejezetben tárgyalt eszközök hogyan kapcsolódnak egymáshoz, illetve a külvilággal hogyan kommunikálnak, ez látható a 5.6 ábrán Processzor adatokat a memóriából egy érvényes memória cím megküldésével tud lekérni, amit a memória az adat buszon elküld a processzornak. Ez gyakorlatban úgynézki, hogy a processzor kirakja az érvényes memória címet a címbuszra és a control buszon keresztül szól a memóriának, majd a memória ki idő elteltével (ez az adat hozzáférés ideje) kirakja a két irányú adat busz a kért adatot, amit a processzor beolvass. Adat írásnál a processzor a cím után, az adatot is kirakja az cím- illetve adatbuszokra. Egy harmadik szereplő is van a képben, ez pedig az bekimeneti eszközöket összefogó I/O vezérlő, aki szintén rajta van az adat és cím buszon, de elsődlegesen a memóriával kommunikál. Ezt a kommunikációt megtudja valósítani az úgynevezett direkt me-

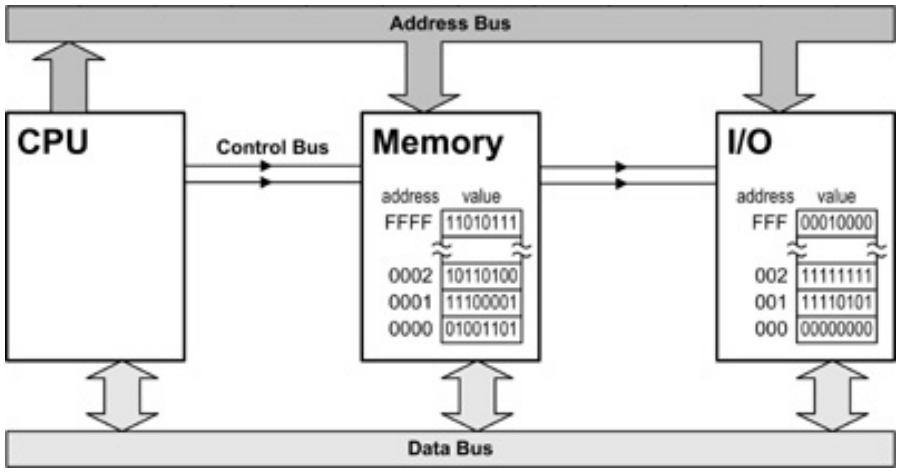


5.4. ábra. Szegmens regiszterek és kapcsolatuk a fizikai memóriával Forrás: IA-32 Architectures Software Developer Manuals

mória hozzáféréssel (DMA) is, amikor is az I/O kizárolagosan írja vagy olvassa memóriát. Ezen idő alatt a processzor nem fér hozzá a memóriához, ezért is fontos, hogy jó sok adatot és utasítást előre olvasson az adott programból, amit végre kell hajtani.



5.5. ábra. Egy egyszerű pipeline felépítése, a különböző színek a különböző utasítás feldolgozó funckiókat jelölik. Forrás: http://lwn.net/Articles/250967/



5.6. ábra. A CPU, memória és az I/O vezérlő blokkdiagramja Forrás: http://www.talktoanit.com/A+/aplus-website/lessons.html

6. fejezet

Hálózatok

Uhlár László

6.1. Egy kis történelem

6.1.1. A kezdetek

Az igény, hogy a számítógépek egymással valamiféle összeköttetésben legyenek, szinte egy idős az első elektronikus számítógépekkel. Kezdetben önálló, szinte egész termeket kitöltő számítógépeken dolgoztak az emberek. Korán megjelent az igény, hogy az egyik gépen megtalálható adat, program minél könnyebben átvihető legyen egy másik gépre anélkül, hogy ehhez külső adathordozót kelljen igénybe venni. A számítógépek méretének és árának csökkenésével egyre inkább elterjedt az a modell, hogy nem egy hatalmas gépen dolgoztak a felhasználók, hanem több kisebb számítógép volt például egy cég iroda házában. Mivel fizikailag egymáshoz közel voltak, jogos igény volt, hogy a viszonylag ritkán használt de drága perifériákból ne kelljen minden géphez külön-külön beszerezni egy példányt (pl.: nyomtató), hanem közösen használhassanak egy ilyen eszközt.
Tehát a számítógépes hálózatok létrehozásának célja:

- Lehetővé teszi az erőforrások megosztását. A rendszerben levő erőforrások (tárolók, nyomtatók) a jogosultságtól függően elérhetők bárki számára.
- Nagyobb megbízhatóságú működést eredményez, hogy az adatok egyszerre több helyen is tárolhatók, az egyik példány megsemmisülése nem okoz adatvesztést. Az azonos funkciójú elemek helyettesíthetik egymást. (Több nyomtató közül választhatunk)
- Gazdaságosan növelhető a teljesítmény. A feladatok egy nagyszámítógép helyett megoszthatók több kisebb teljesítményű eszköz között.
- Elérhetővé válnak a központi adatbázisok. Ezek az adatbázisok sok helyről lekérdezhetők, és sok helyről tölthetők. Csak így képzelhető el pl. egy valóban aktuális raktár vagy megrendelés állomány kezelés egy nagyvállalatnál.
- A hálózati rendszer kommunikációs közegként is használható.

6.1.2. Az ARPA project

Az 1960-as évek közepén (dúl a hidegháború) az amerikai Védelmi Minisztérium (U. S. Department of Defense) olyan parancsközlő hálózat kialakítását tűzte ki célul, mely átvészel egy esetleges atomcsapást. A fejlesztéseket a minisztérium ösztöndíjakkal támogatta. Az elméleti kutatások után olyan hálózat kialakítására írtak ki pályázatot, amely csomóponti gépekből(IMP, Interface Management Processor) áll, amelyeket adathálózat köt össze és néhány IMP megsemmisülése esetén is működőképes marad a hálózat többi része. A tenderre több cég is nevezett, a győztes 1969-ben állította üzembe az első csomópontot, 1972-re 37-re nőtt a csomópontok száma. Ekkoriban kapta az ARPAnet nevet ez a hálózat (Advanced Research Project Agency). A 70-es évek

végére összeköttetések épültek ki más helyi hálózatok és az ARPAnet között, mára ez a hálózat behálózza az egész Földet. A 80-as évektől nevezik a hálózatok ezen hálózatát internetnek.

6.2. „Rétegek”

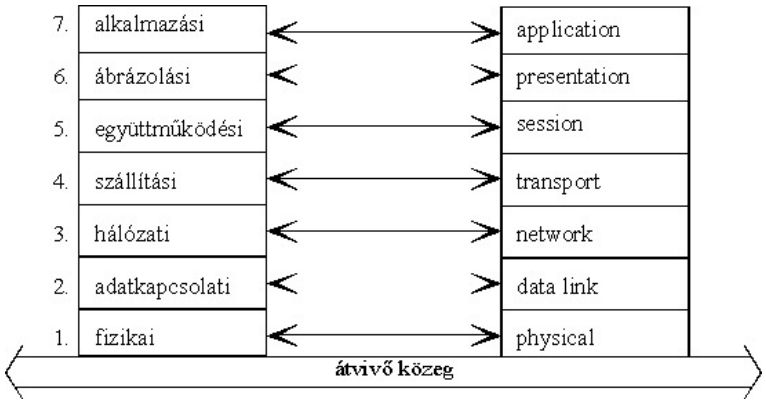
6.2.1. Mi ez?

Bizonyára el tudjuk képzelni, hogy a fentebb vázolt hálózatokon a kommunikáció meglehetősen összetett és bonyolult dolog. Nem lenne szerencsés, ha a programozónak olyan hálózati kommunikációra képes programokat kellene írnia, amely a teljes kommunikáció minden aspektusát megoldja. Egy program kellene, hogy gondoskodjon a megfelelő feszültségszintek előállításától kezdve a célzott gép azonosításáig mindenről. Ha a hálózati kábelezésen változtatnak és a réz vezeték helyett üvegszálat használnának, az egész programot újra kellene írni, ki kellene vserélni. Ezért kialakult az a fajta modellezése a hálózati kommunikációnak, amikor logikailag több részre bontják a folyamatot, az egyes részek a folyamat egy jól meghatározott részéért felelnek, azt kell megvalósítaniuk. Csak arról kell gondoskodni, hogy az egyes részek (rétegek) egymást megértsék, egy jól definiált interfészt kell egymás felé mutatniuk. például a postai levelezésnél a postaládát kiürítő dolgozónak nem kell tudnia repülőt vezetni vagy hajót építeni és átszelni az óceánt, ha oda szól a levél, neki csak a ládát kell tudnia kiüríteni (de azt hiba nélkül) és el kell juttatnia a borítékokat a megfelelő helyre. Az ottani dolgozónak pedig nem kell tudnia, hogy a város mely részén vannak ürítendő postaládák és azokat hogyan kell kinyitni, neki csak a küldeményeket kell bizonyos szempontok szerint szétválogatnia, stb.
Hasonló módon az egyes rétegek szolgáltatásait implementáló programozóknak sem kell az egész kommunikációs problémát egyben vizsgálniuk, nekik elég csak az adott rétegre koncentrálniuk.

6.2.2. ISO OSI

A Nemzetközi Szabványügyi Szervezet (ISO, International Standard Organization) létrehozott egy ajánlást (nem szabvány!!!), amelyet OSI (Open System Interconnect - nyílt rendszerek összekapcsolása) modellnek hívnak, és amely hét rétegre bontja logikailag a számítógép hálózatok működését.

Segítheti a megértést, ha két magasrangú államférfira gondolunk, akik tolmácsok segítségével



6.1. ábra. Az OSI rétegek

kommunikálnak egymással: az egyik vezető a saját tolmácsának mondja, az elmondja a másik tolmácsnak, az lefordítja a saját főnökének. Kik kommunikálnak egymással: a vezetők beszélni a saját tolmácsaikkal beszélnek, de mégis a két államférfi cserél eszmét a beszélgetés sorá. A számítógép-hálózatokon folyó kommunikáció során is az egyes egymásnak megfelelő rétegek kommunikálnak logikailag, csak ennek során kihasználják az alattuk lévő réteg szolgáltatásait, amely esetleg szintén kihasználja az alatta lévő rétegét, stb. A tényleges fizikai jeltovábbítás a fizikai rétegen történik.

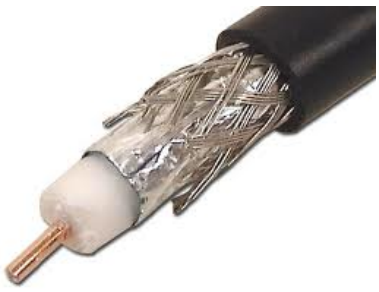
6.3. Az egyes rétegek feladata

6.3.1. A fizikai réteg

A tényleges fizikai jeltovábbítás terepe. Sok fontos dolgot kell ennek a rétegnek megoldania. A számítástechnikában általánosan használt digitális jeleket kell valahogyan az adott közegen továbbítani illetve fogadni.
Itt említendő fontos fogalom a baud. Értéke megmutatja a másodpercenkénti jelváltások számát az adott közegen. Ez nem feltétlenül egyezik meg a másodpercenként átküldött bitek számával, hiszen ha egy jelváltás valamilyen alkalmas tömörítő kódolással négy bit információt tud hordozni, akkor egy 1000 baud-os vonal 4000 bitet visz át másodpercenként.
Másik fontos fogalom az átviteli sebesség. Mértékegysége a bit per secundum (bps): 10 bps a sebesség, ha 10 bit információ továbbítódik egy másodperc alatt. Elterjedt prefixált mértékegységek még a kbps: 256 kbps a sebesség, ha 256 ezer bit információ továbbítódik másodpercenként, az Mbps illetve a Gbps hasonlóan megabit illetve gigabit egységekben. Nem összekeverendő a MBps-el, ami megabájtban adja meg az átvitt adatmennyiséget.
Szintén megemlítendő fogalom a szinkronizáció: egy megfelelő jelszint 1 másodpercen keresztül értelmezhető 128000 db egyesnek (128kbps vonal), de értelmezhető 256000 db egyesnek is (256 kbps vonal).
Gyakori, hogy a közeg analóg jelek továbbítására képes (analóg: valamilyen fizikai jellemző nagysága hordozza az információt), ekkor meg kell oldani a digitális jelek (valamely fizikai jellemző léte vagy nem léte hordozza a jelet) analóg jellé konvetálását(DAC: digitális analóg konverter), majd a vonal túlvégén az érkező analóg jeleket „digitalizálni” kell (ADC: analóg digitális konverter).
Az analóg-digitális konverzióval sok helyen találkozhatunk (például hang digitalizálás). Aki utána szeretne jobban olvasni, ajánlható a következő kis leírás: http://studio.pataky.hu/edu/14p/atvitel/PCM.pdf. Érdemes lehet utánajárni a PCM, DSD, PWM rövidítések jelentésének is.

A „vezeték”

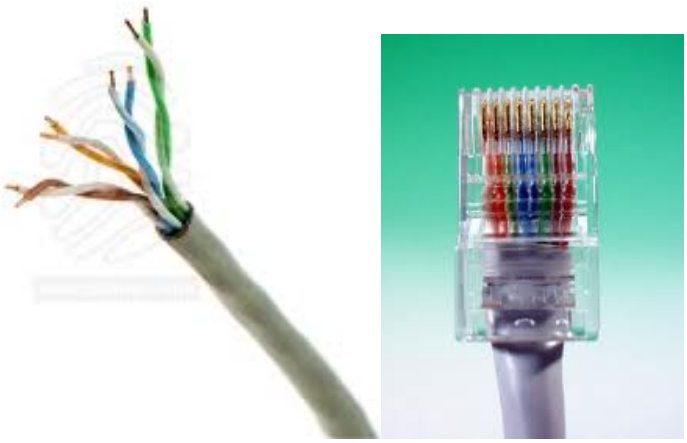
–A koaxiális kábel olyan, a híradástechnikában használt vezetéktípus, amely egy belső vezető érből, dielektrikumból, fémhálóból és külső szigetelésből áll. A fémháló szerepe az árnyékolás, azaz a belső éren továbbított jelek megóvása a külső zavaroktól. Elsősorban rádiófrekvenciás jelek továbbítására használják.



6.2. ábra. Koaxiális kábel

–UTP (Unshielded Twisted Pair: árnyékolatlan csavart érpár). Felhasználóként ezzel a kábellel találkozunk napjainkban a legtöbbet. 8 szál vezeték párosával egymás köré tekerve alkotja a kábelt, a koaxszal ellentétben nincs külön árnyékolása. Az egymás köré tekeréssel csökkentik a környezet zavaró jeleit. A csatlakozóban a kábelek sorrendje fontos, nem mindegy, hogy mely színű vezeték hová kerül. Készülhet egyenes bekötésű illetve „cross link” kábel is. A mai modern eszközök automatikusan érzékelik, hogy milyen kábelt csatlakoztattunk és ennek megfelelően veszik használatba.
Az UTP kábeleket több kategóriába sorolják a paramétereik szerint. Ezek:

- CAT1 - telefonkábel (hangátvitel, 2 érpár)



(a) UTP kábel (b) UTP csatlakozó

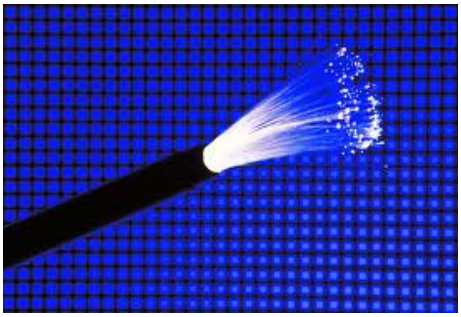
6.3. ábra. A „csavart érpár”

- CAT2 - maximum 4 Mb/s adatátviteli sebesség érhető el vele.
- CAT3 - 10 Mb/s az adatátviteli sebessége. Csillag topológiánál alkalmazzák, ethernet hálózatokban (Legacy Ethernet[10MB/s-os] közege).
- CAT4 - max. 20 Mb/s adatátviteli sebességű.
- CAT5 - 100 Mb/s adatátviteli sebességű, csillag topológiánál alkalmazzák, ethernet hálózatokban.
- CAT5e, CAT6 - 1000 Mb/s átviteli sebesség.

–A jelenlegi legkorszerűbb vezetékes adatátviteli módszer az üvegszál vagy más néven optikai technológia alkalmazása. Üvegszálas hálózat kiépítésére akkor kerül sor, ha különösen nagy elektromágneses hatások érik a vezetékeket vagy nagy távolságokat kell áthidalni. Itt a fényátvezető anyagból készült optikai szálon tovahaladó fényimpulzusok szállítják a jeleket. Az optikai kábel egy olyan vezeték, amelynek közepén üvegszál fut. Ezt az üvegszálat gondosan kiválasztott anyagú burkolat veszi körül. A különleges anyag tulajdonsága, hogy az ide-oda cikázó fény sohasem tudja elhagyni a kábelt. Ezért a fény a vezeték elején lép be és a végén lép ki belőle. De így is meg kell erősíteni és újra kell rendezni a fényt. A legnagyobb áthidalható távolság manapság 80 kilométer, ami lényegesen hosszabb táv a hasonló rendű kábelekhez képest. Az adó, ami lehet LED vagy lézer, elektronikus adatot küld át a kábelen melyet előzőleg fotonná alakítottak. A fotonok hullámhosszai az 1200-1500-ig terjedő nanométer spektrumban lehetnek.
Az optikai átviteli rendszer három komponensből áll: az átviteli közeg-ből (hajszálvékony üveg vagy szilikát), amit egy szilárd fénytörő réteg véd (szintén üveg vagy műanyag), a fényforrásból (LED vagy lézerdióda) és a fényérzékelő-ből (fotodióda).
–WIFI: vezeték nélküli mikrohullámú kommunikációt megvalósító szabvány (IEEE802.11). Napjainkban a mobil eszkzök (laptop) elterjedésével egyre népszerűbb a felhasználók körében. A kábeles kommunikációnál nagyobb hangsúlyt kapnak a terepviszonyok, távolság, lehallgathatóság.

6.3.2. Az adatkapcsolati réteg

Az adatkapcsolati réteg (Data Link Layer) alapvető feladata, hogy egy bitfolyam átvitelére képes fizikai rendszert egy olyan eszközzé alakítsa, ami adatátviteli hibáktól mentes szolgáltatást nyújt a hálózati réteg számára. Az ellenőrzés érdekében az adó oldal a bemenő adatokat meghatározott hosszúság darabokra – keretekké- tördeli. Az adatkerteket fel kell ismernünk. Ezért a



6.4. ábra. Üvegszál

keretek elé és mögé speciális bitmintákat helyezünk el. A bitmintáink az adatkeret belsejében is előfordulhatnak, ezért ellenőrző eljárásokat kell kitalálnunk. Az adatkapcsolati rétegnek kell feldolgozni az ellenállomásról érkező nyugta - kereteket is. Fontos feladata az adatfolyam vezérlés (flow controll), amelyben a vevőhöz igazítja az adás idejét és sebességét. Felépíti és lebontja a kapcsolatokat, a fogadott adatkereteket a megfelelő fejrész levágása után továbbítja a kijelölt folyamatnak. Fizikai cím (MAC) segítségével azonosítja, hogy mely hálózati interfészen (NIC) kell továbbítani az adatot.

Az ethernet

Több technológia is kialakult a fentiek megoldására, környezetünkben legelterjedtebben az ethernettel találkozhatunk (IEEE 802.3 szabvány). A többiekhez hasonlóan az adatküldésre készülő állomás is „hallja" a közegen zajló forgalomat (Multiple Access). Amennyiben azt érzékeli, hogy már valaki adás állapotban van, azaz vivőt érzékel (Carrier Sense), akkor várakozik az adás megszűnésére. Amikor a közegen később „csend" lesz, akkor megkezdi az adást, amelyet azok az állomások vesznek, amelyek az üzenetben szereplő (MAC) cím alapján megszólítva érzik magukat. Előfordulhat olyan helyzet, hogy (megközelítően) egyszerre többen kezdenek adásba, ekkor az adók egymást zavarják, ezért a közegre került információ sérül. A hálózat ilyen állapotát (ütközést) detektálni kell (Collision Detection). Ezt az állapotot az adók az adott jel visszaolvasásakor érzékelt hibákról ismerik fel („nem hallja tisztán a saját hangját"). Ekkor minden, az ütközést érzékelő adó beszünteti az adást, majd véletlenszerű késleltetési idő után újból próbálkozik az adással. A véletlen időzítést követően valószínű, hogy az egyik adó kellően korán lefoglalja a közeget és a többiek az adásuk megkezdése előtt érzékelik azt.
–Kezdetben egy sínszerűen kialakított koaxiális kábellel megvalósított vezetékre voltak felfűzve a hálózat gépei. Mindenki mindenki adását hallotta, sok lehetett így az ütközés. A technológia miatt rendkívül érzékeny volt a kábel szakadásokra: bárholt megszakadt a vezeték, az egész sínen megállt a forgalom. A sok ütközés elkerülésére jelenthetett megoldást, ha bridge vagy router szerű eszközzel (lásd később) két vagy több „ütközési tartományra" osztották. Egy-egy tartományon belül a szakadás továbbra is teljes leállást okozott.
–Ennek megoldására találták ki a csillag szerűen egy központi „elosztóba" csatlakozó kábelekkel megvalósított úgynevezett csillag topológiát. Ez a központi eszköz régebben hub volt, ma már inkább switch. Kábelszakadás esetén csak az az egy gép esett ki a kommunikációból. A hub a fizikai rétegben „dolgozik", az egyik csatlakozóján beérkező jelet az összes csatlakozón kiküldi felerősítve (repeater). Ezzel a megoldással az ütközések száma nem csökkent, továbbra is egy ütközési tarttományt alkotnak a gépek. A switch erre is megoldást kínál. A sweitch (kapcsoló) a keretekben található fizikai cím (MAC) alapján dönti el, hogy mely csatlakozón küldi tovább az adatot. A kapcsoló működése egy dinamikusan karbantartott táblázatra épül, amelyben a kapcsoló minden portjához feljegyzi az adott porton beérkező keretek küldőjének MAC címét. Ezzel a kapcsoló megismeri a hozzá kapcsolódó gépek helyzetét, tehát azt, hogy az egyes gépek a kapcsoló melyik interfészéhez kapcsolódnak. Egy beérkezett keret továbbításához csak meg kell vizsgálnia a táblázatot, hogy a keretben szereplő cél MAC cím melyik interfészén keresztül érhető el. Egy interfészhez több gép MAC címe is feljegyezhető, ezért nincs akadálya hub-switch

vagy switch-switch kapcsolatnak sem. A switch bekapcsolásakor kezdődő tanulási folyamat során fokozatosan alakul ki a kapcsoló táblázata. Ezért normál jelenségnek tekinthető, ha egy olyan keretet kell továbbítania, amelynek a címzettje még a switch számára ismeretlen irányban van. Ekkor az ún. elárasztást alkalmazza, azaz a beérkezett keretet a fogadó port kivételével az összes többi portján kiküldi.

6.3.3. A hálózati réteg

A hálózati réteg feladata a csomagok eljuttatása a forrástól a célig. A célig egy csomag valószínûleg több csomópontot is érint. Ehhez természetesen ismerni kell az átviteli hálózat felépítését, azaz a topológiáját, és ki kell választania a valamilyen szempontból optimális útvonalat. Ha a forrás és a cél eltérõ típusú hálózatokban vannak, a réteg feladata a hálózatok közti különbségbõl adódó problémák megoldása. A hálózati réteg tervezésénél az egyik legnagyobb probléma az összeköttetésre módjának meghatározása. Összeköttetés alapú hálózatban forrás és a cél között felépült állandó úton vándorolnak a csomagok gondoljunk a vezetékes telefonálásra, mint analógiára). Összeköttetés mentes hálózatban elvileg minden egyes csomag különbözõ útvonalakat követhet, mivel a csomagok útválasztása egymástól független. Ilyenkor a csomagoknak tartalmazniuk kell mind a forrás, mind a cél teljes címét.

IP címek

Ahhoz, hogy a hálózati réteg megtalálja a csomagok címzettjét, minden gépnek rendelkeznie kell egy egyedi címmel, ez az IP cím. A jelenleg (még) legelterjedtebb IPv4 szerint ez a cím egy 32 jegyű bináris szám, amit a jobb olvashatóság miatt 8 bitenként decimális számmá alakítunk, így szoktuk látni ezeket (pontozott négyes jelölés). Ilyen például: 193.224.69.67. A számítógépeinket fizikai vagy logikai szempontok szerint alhálózatba sorolhatjuk. Az egy alhálózatban lévő gépek egymással közvetlenül (útválasztók nélkül) tudnak kommunikálni. Hogy két gép egy alhálózatban található-e, az az IP címéből kell, hogy kiderüljön. Kezdetben a lehetséges címeket osztályokba sorolták, így beszélhetünk A, B, C, stb. osztályú címekről.
–A osztály: az első számjegy (a 32 -ből) 0, utána lévő 7 bit azonosítja a hálózatot, a maradék 24 bit a hálózaton belül az egyes hosztokat. A legkisebb ilyen hálózat azonosító az 1 lehet, a legnagyobb a 127. (a csupa nullát tiltjuk). Egy-egy ilyen hálózatban rengeteg különböző számítógép lehet: kb.: 2^24.
–B osztály: a cím 10-val kezdődik, 16 bit azonosítja a hálózatot, azon belül a maradék 16 bit a számítógépet. Így a legkisebb hálózatszám a 128.0 lehet, a legnagyobb a 191.255.
–C osztály:a cím 110-val kezdődik, 24 bit azonosítja a hálózatot, a maradék 8 a hostokat. A legkisebb ilyen hálózatcím a 192.0.0, a legnagyobb a 223.255.255.
A többi cím nincs kiosztva, speciális célokra fenntartottak.
Minden osztályban kijelöltek olyan címtartományt, amelyek a nyílt interneten nem használhatók (nem „publikus”). Ezeket a privát címeket egy-egy belső, az internettől elzárt alhálózatban lehet használni. Azaz ilyen privát IP címmel lehet, hogy egy időben több számítógép is rendelkezik. Ezek:
–A osztályban: 10.0.0.0, illetve a 127.0.0.0 a visszacsatoló interfész számára fenntartva.
–B osztályban: 172.16.0.0-172.31.0.0.
–C osztályban: 192.168.1.0-192.168.255.0.
Tehát ezeket a címeket csak belső hálózatokban (intranetekben) lehet használni.
Kezdetben az egyes hálózati eszközök tehát az IP címből meg tudták állapítani, hogy két gép egy alhálózaton van-e.(alapértelmezett hálózatszám). Viszont már elég korán felismerték annak a veszélyét, hogy így nem túl gazdaságos a címek kiosztása, pl.: ha valaki megszerzett egy A osztályú címosztályt, akkor 2^24 darab különböző gépet helyezhetne el benne, de ennyire valószínűleg nincs szükség. Felmerült az igény arra, hogy ezek helyett az alapértelmezett hálózatszámok helyett szabadabban lehessen gazdálkodni a még meglevő címekkel. továbbra is beszélhetünk A, B, stb. osztályú címekről, de már nem magától értetődő, hogy hány bit azonosítja az alhálózatot.

Az alhálózati maszk

Az IP cím mellé megadunk egy újabb 32 jegyű (bites) számot, amely az elején csupa 1-t tartalmaz, utána csupa 0-t. Ezt szintén pontozott négyes jelöléssel. Ilyen például: 255.255.192.0. Sokkal kényelmesebb leírást, használhatóságot biztosít az úgynevezett CIDR (Classless Interdomain Routing) jelölés, amely esetén a pontozott négyes jelölés helyett az elhálózati maszk egyeseinek a számát adják meg. Az előző példával élve 18 jegy azonosítja a hálózatot. (példa később)
Érdemes lehet kicsit eljátszani az alábbi oldalon: http://www.fport.hu/index.php?site=cidr

Példák

1. példa
Legyen egy gép (A gép) IP címe 193.224.69.67, alhálózati maszkja: 255.255.255.192, másként: 193.224.69.67/26. Binárisan:

11000001111000000100010101000011	IP cím
11111111111111111111111111000000	alhálózati maszk

Szeretne kommunikálni a 193.224.69.121 című géppel (B gép). Ehhez először el kell döntenie, hogy egy alhálózatban vannak-e? A fenti táblázatban lévő IP címnek tehát veszi a hálózat azonosító bitjeit (első 26 jegy), azaz képezi a két számmal a logikai ÉS (AND) műveletet. Így az A gép megkapja, hogy az ő hálózat száma:
11000001111000000100010101
A címzett gép (B) címéből szintén veszi az első 26 jegyet, megnézi, hogy egyeznek-e? Igen, tehát egy alhálózatban vannak, közvetlenül kommunikálhat vele.
Ha ethernet hálózatról van szó, akkor az adatok tényleges küldéséhez szükség van a címzett kártyájának MAC (Media Access Control)címére. (Ez az egész világon elvileg egyedi szám, 12 jegyű hexadecimális szám formájában szokták megadni). Hogy egy IP címhez milyen MAC cím tartozik, ennek kiderítése az ARP feladata (Address Resolution Protokol)
Az ARP egy belső táblázatának (ARP cache) adatait használja az IP címhez tartozó MAC cím meghatározásakor. Ha a kérdéses MAC cím nem szerepel az ARP táblában, akkor egy olyan üzenetet küld ki a hálózatra, amelyet minden hálókártya elfogad, mint neki szóló üzenet (broadcast). Az üzenet tartalmazza a címzett IP címet. Amely gép felismeri a saját IP címét, az egy célzott, már csak a feladónak szóló üzenetben válaszol, elküldi, hogy mi a MAC címe. Ezt a feladó elraktározza az ARP táblában, majd elkezdődhet a kommunikáció. Fordított működést valósít meg a RARP (Reverse ARP): MAC cím alapján próbálja kideríteni az IP címet. Az ARP cache tartalmát lekérdezhetjük az arp paranccsal (root jog!!!). Nézzük meg a man oldalát!

2. példa
A kommunikáció kezdeményezője legyen megint a fenti példában szereplő 193.224.69.67/26 című gép. Kezdeményezzen kommunikációt a 193.224.69.51 IP című géppel. Egy alhálóban vannak? Ehhez a címzett címének veszi az első 26 jegyét: 11000001111000000100010100. Látható, hogy különböznek, azaz nem egy alhálóban vannak. Ekkor egy routernek (útvonal választónak) kell továbbítania az adatokat, annak a dolga valahogyan azokat a címzetthez eljuttatni. Minden hálózatba kötött gépnek van routing táblája, amely arról tartalmaz információkat, hogy bizonyos típusú IP címek esetén mely hálózati csatolón (hálókártyán) küldje ki az adatokat. Ennek tartalmaznia kell egy default bejegyzést: amely címről nincs bejegyzés, hogy merre küldjük, azt küldjük erre tovább. Ez a routing tábla lekérdezhető a route utasítással (root jog!!!). Nézzük meg a man oldalát!
Innét már megegyezik a folyamat a fenti példával: kell az alapértelmezett átjáró (default router) MAC címe. Ha benne van az ARP cache-ben, akkor...., különben....

Mindez linuxon

Az egyes interfészeket beállítani az ifconfig utasítással lehet (root jog!!!) Ha minden kapcsoló és paraméter nélkül adjuk ki az ifconfig utasítást, akkor a lentihez hasonló, az adott interfészről adatokat közlő listát kaphatunk.

```
wlan0     Link encap:Ethernet  HWaddr 90:4c:e5:c9:5d:30
```

```
inet addr:192.168.1.100  Bcast:192.168.1.255  Mask:255.255.255.0
inet6 addr: fe80::924c:e5ff:fec9:5d30/64 Scope:Link
UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
RX packets:460101 errors:0 dropped:0 overruns:0 frame:0
TX packets:310566 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:591973817 (564.5 MiB)  TX bytes:41034374 (39.1 MiB)
```

Gondosan tanulmányozzuk a manualt!
Sajnos az egyes beállítások megváltoztatásához root jog kell, ezért ezeket kipróbálni nem tudjuk.
A gép bekapcsolásakor betöltődő beállításokat GNU/Debian alatt a /etc/network/interfaces fájl tartalmazza. Listázzuk ki a tartalmát! (cat, manual!!!)
Fontos információkhoz juthatunk a traceroute parancs segítségével. Írjuk be: `traceroute 193.224.69.67` . Az adott géphez vezető „úton" útbaeső csomópontok néhány adatát láthatjuk. (manual!!!)

A DHCP

A fentiekből látható, hogy elég bonyolult egy hálózatban az IP címek karbantartása. ki milyet, milyen maszkkal, ne legyen ütközés, stb. Ennek a problémának a megoldására dolgozták ki a DHCP-t. (Dynamic Host Configuration Protokol). Többek között a fenti (IP cím) beállítások automatikus megoldását teszi lehetővé, ezzel segítve a rendszergazda, a felhasználó munkáját.

A DNS

Igen ám, de ha mi beírunk a böngészőbe egy címet, az a legritkább esetben pontozott négyes jelölés, legtöbbször értelmes emberi kifejezések. Most akkor a gép számokkal azonosítja a másikat vagy „emberi" nevekkel? Számokkal!
Hogyan kapja meg a hálózati réteg a cím alapján az IP címet? Erre szolgál a DNS.
Egy világméretű elosztott (nem egy gépen tárolt, hanem részenként szétosztott) adatbázisban találhatóak a névhez IP címet rendelő információk. Ha a gépünknek szüksége van a turdus.itk.ppke.hu gép címére, egy kérdést küld a .hu legfelső szintű (top level) domain valamely DNS szerveréhez (több gépen vannak ugyan azok az adatok), hogy mely gép tárolja a ppke.hu tartomány címeit. Ekkor visszakapja a gépünk az egyik ilyen DNS szerver címét. Egy második körben már ehhez fordul egy újabb kéréssel, hogy megtudja, hogy mely gép tárolja az itk.ppke.hu tartomány címeit. Ekkor innét is visszakapja egy olyan DNS szerver címét, amelyhez fordulva kéréssel, már megkaphatja a turdus.itk.ppke.hu gép IP címét. (A kapott eredményt egy ideig megőrzi, legközelebb ne kelljen az egész utat újra bejárni.)
Próbáljuk ki! Egy böngésző címsorába írjuk be: 217.20.138.26! Ezek talán nehezebben megjegyezhetőek, mint az „emberi" elnevezések.

Az IPv6

Napjainkra égető problémává vált, hogy a fentebb vázolt 32 bites IP címzés nem elegendő, elfogynak a kioszthtató címek. Erre kínál megoldást az IPv6, amely esetén 128 bites címe van minden hálózati csatolónak.

A NAT

Az IPv4-es címek elégtelen volta miatt (nem jut minden gépnek egyedi) dolgozták ki a Hálózati Címfordítást (Network Address Translation), amely esetén az alhálózatunkon használhatunk privát címeket (melyeknek csak az alhálózatunkon kell egyedinek lennie!). Ha a világhálóra akarunk csatlakozni, akkor egy „átjáró" gépen keresztül jutnak ki a csomagjaink, amely gépen futó NAT szolgáltatás privát címeinket kicseréli az ő egyetlen publikus IP címére és így küldi tovább azokat, közben egy táblázatban feljegyzi, hogy mely privát IP-t írta át. Ha jön a válasz, ő kapja, hiszen ez a gép szerepel feladóként, a táblázata alapján továbbküldi a választ a belső hálóba a ténylegesen kommunikáló privát című gépnek.

6.3.4. A szállítási réteg

A szállítási réteg az OSI modell legbonyolultabb rétege. Fő célja, hogy megbízható, gazdaságos szolgálatot nyújtson a felette lévő rétegeknek. A szállítási réteg transzparens a felső rétegek számára.
A szállítási réteg tipikus feladatai:

- forgalom szabályozás
- multiplexelés
- virtuális áramkörök felügyelete
- hibajavítás
- csomagképzés és csomagok visszaállítása a felsőbb rétegek számára.

A forgalom szabályozás feladata, hogy az adó ne adjon több adatot, mint amit a vevő fogadni tud. A multiplexelés lehetővé teszi, hogy több alkalmazás használja ugyanazt a fizikai összeköttetést. Az összeköttetések nyalábolása pedig lehetővé teszi, hogy egy alkalmazás használjon több összeköttetést egy időben.

6.3.5. Az együttműködési réteg

A réteg lehetővé teszi, hogy két számítógép felhasználói kapcsolatot létesítsenek egymással. A viszonyréteg segítségével egy felhasználó állományokat mozgathat számítógépek között. Jellegzetes feladata a logikai kapcsolat felépítése és bontása, párbeszéd szervezése. Szinkronizációs feladatokat is ellát, ellenőrzési pontok beépítésével.

6.3.6. A megjelenítési réteg

A réteg a viszonyrétegen felül helyezkedik el, és olyan szolgáltatásokat ad, amelyekre a legtöbb alkalmazói programnak szüksége van, amikor a hálózatot használja. Ez a réteg foglalkozik a hálózaton továbbítandó adatok ábrázolásával: el kell döntenie, hogy milyen egységes struktúrába szervezze az adatokat, amelyeket a felette elhelyezkedõ alkalmazói rétegtõl kap. A legtöbb program például neveket, számokat, stb. küld egymásnak, amelyeket esetenként bonyolult adatszerkezetekként ábrázolnak. Ehhez jön még az a tény, hogy a különböző számítógépek különböző kódolásokat alkalmaznak (ASCII,...). Annak érdekében, hogy a számítógépek egymással kommunikálni tudjanak, az adatokat a hálózaton egységes szabvány szerint kell bitek egymásutánjára kódolni. Ezt végzi el a megjelenítési réteg. Egyéb feladatai közé tartozhat még az adattömörítés, illetve a titkosítás is.

6.3.7. Az alkalmazási réteg

Széles körben igényelt protokollokat tartalmaz. Az állománytovábbításokon kívül ehhez a réteghez tartozik még az elektronikus levelezés, a távoli munkabevitel, a katalóguskikeresés, és még egy sor egyéb, általános-, ill. speciális célú alkalmazási feladat is. Ez a réteg kapcsolódik szorosan a felhasználóhoz, itt kell a hálózati felhasználói kapcsolatok megoldásait megvalósítani. Ezeken kívül definiál egy hálózati virtuális terminált amely segít az eltérő hálózati terminálok kezelésében.(Azok a szoftverek, melyeket már ténylegesen a felhasználó használ)

6.4. A TCP/IP

6.4.1. A protokoll

Az informatikában a protokoll egy egyezmény, vagy szabvány, amely leírja, hogy a hálózat résztvevői miképp tudnak egymással kommunikálni. Ez többnyire a kapcsolat felvételét, kommunikációt, adat továbbítást jelent. Gyakorlati szempontból a protokoll azt mondja meg, hogy milyen sorrendben milyen protokoll-üzeneteket küldhetnek egymásnak a csomópontok, illetve az üzenetek pontos felépítését, az abban szereplő adatok jelentését is megadja.

6.4.2. A port szám

Egy adott gépen több olyan program is futhat, amely hálózati kommunikációra képes. Ha egy gép „megszólít” egy másikat, a címzett honnét fogja tudni, hogy mely programjával akarunk kommunikálni. A portszám valójában ezeket azonosítja. Képzeljünk el egy bankot, ahová egy ajtón lehet bejutni (Hálókártya), de benn több ablak van, számokkal megkülönböztetve, mindegyik mögött (már amelyik nyitva van) mással foglakozó munkatárs ül: pénztár, hitel, stb. Így van ez az informatikában is. Adjuk ki a következő utasítást! `cat /etc/services`
A kapott lista azokat a program - portszám összerendeléseket tatalmazza, amelyek a leggyakoribbak.

6.4.3. A TCP

Az üzenetek széttördelését, összeállítását, az elveszett részek újraadását, a datagrammok helyes sorrendjének visszaállítását mind a TCP (transmission control protocol – átvitelvezérlési protokoll) végzi. Az egyes datagrammok útvonalának a meghatározását (routing) az IP (internet protocol) hajtja végre. Mindez azt a látszatot kelti, hogy a munka tetemes része a TCP-re hárul. Kis kiterjedésû hálózatokban ez így is van, azonban az Interneten egy datagrammnak a rendeltetési helyre való juttatása igen összetett feladatot jelenthet. Egy datagramm több hálózaton mehet keresztül míg végül eljut a célállomásra. Például a Rutgers Egyetemről kiindulva a John von Neumann Supercomputing Center-ig soros vonalon keresztül, majd onnan (egy pár Ethernet hálózaton átjutva) 56Kbaud telefonvonalakon keresztül jut el egy másik NSFnet hálózatra stb...
A különbözõ átviteli közegekbõl adódó inkompatibilitások kezelése és a célállomásokhoz vezetõ útvonalak végigkövetése komplex feladat. Meg kell jegyezni azonban, hogy a TCP és az IP közti interfész rendkívül egyszerû: a TCP egy datagrammot ad át az IP-nek egy rendeltetési címmel együtt. Az IP semmit sem tud arról, hogy ez az információ hogyan viszonyul más datagrammokhoz.
Az alábbiakban a tipikus TCP/IP hálózaton keresztül haladó üzenetre rárakódó fejléceket tekintjük át:
Kezdetnek vegyünk egy egyszerû adatfolyamot (pl. egy állomány tartalma), amelyet egy másik számítógépnek szeretnénk elküldeni:

.....

Ezt a TCP megcsonkítja. (Ennek érdekében tudatni kell a protokollal, hogy mekkora az a maximális adatméret, amelyet az adott hálózat még kezelni tud. Valójában az összeköttetés két végén a TCP-k közlik egymással az általuk kezelhetõ maximális méretet, majd veszik a kisebbiket.)

....
Minden datagramm elé egy TCP fejléc kerül, amely legalább 20 oktetből áll. Ezek közül a legfontosabbak: egy forrás- és egy célport, valamint egy sorszám. A portok az összeköttetések végpontjait azonosítják. Tegyük fel például, hogy egyszerre 3 felhasználó továbbít állományokat. A TCP ezekhez az átvitelekhez az 1000, 1001 és 1002 portokat rendelheti. Datagramm küldésekor az allokált port válik a forrásporttá, mivel innen indul ki a datagramm. A kapcsolat másik végénél lévõ TCP szintén hozzárendeli a saját portját az átvitelhez. A küldõ oldali TCP-nek a célport számát is tudnia kell (ezt az információt a kapcsolat felépülésekor szerzi meg), amelyet az a fejléc célport mezőjébe helyez. Ha a másik oldalról érkezik egy datagramm, akkor annak TCP fejlécében a forrás- és a célportok tartalma ellentétes, hiszen ekkor az a forrás, ez pedig a rendeltetési hely. Minden datagrammnak van egy sorszáma, amely a vevő oldalt arról biztosítja, hogy minden adatot helyes sorrendben kapjon meg, és ne veszítsen el egyet se a datagrammok közül.
Ha a TCP fejlécet T-vel jelöljük, akkor az eredeti állományunk így néz ki:
T.... T.... T.... T.... T.... T.... T.... T.... T....

6.4.4. Az IP

A TCP az általa feldolgozott datagrammokat átadja az IP-nek. Persze ezzel együtt közölnie kell a rendeltetési hely Internet címét is. Az IP-t ezeken kívül nem érdekli más: nem számít, hogy mi található a datagrammban vagy, hogy hogyan néz ki a TCP fejléc. Az IP feladata abban áll, hogy a datagramm számára megkeresse a megfelelõ útvonalat és azt a másik oldalhoz eljuttassa. Az útközben fellelhetõ átjárók és egyéb közbülsõ rendszereken való átjutás megkönnyítésére az IP a datagrammhoz hozzáteszi a saját fejlécét. A fejléc fõ részei a forrás, és a rendeltetési hely IP címe, a protokollszám és egy ellenőrző összeg. A forrás címe a küldő gép címét tartalmazza. (Ez azért szükséges, hogy a vevő oldal tudja honnan érkezett az adat.) A rendeltetési hely címe a vevő oldali gép címét jelenti. (Ez pedig azért szükséges, hogy a közbenső átjárók továbbítani tudják az adatot.) A protokollszám kijelöli, hogy a datagramm a különböző szállítási folyamatok közül melyikhez tartozik.
Ha az IP fejlécet I-vel jelöljük, akkor az eredeti állományunk így néz ki:
IT.... IT.... IT.... IT.... IT.... IT.... IT.... IT.... IT....
Ha ethernet hálózatunk van, akkor erre még az ethernet is ráülteti a maga fejlécét, a végére illeszt egy ellenőrző összeget (a hibamentes átvitelt lehet ezzel ellenőrizni). Ha az Ethernet fejlécet E-vel, az ellenõrzõösszeget pedig C-vel jelöljük, akkor az eredeti állományunk így néz ki:
EIT....C EIT....C EIT....C EIT....C EIT....C EIT....C EIT....C
A csomagok megérkezésekor persze a fenti fejlécek mindegyikét leszedi a megfelelõ protokoll. Az ethernet interfész az ethernet fejlécet és az ethernet ellenőrző összeget szedi le. Ezekután ellenőrzi a típuskódot. Mivel az az IP-re mutat, ezért a datagrammot átadja az IP-nek, amely a Protokoll mező tartalmát ellenőrzi. Itt azt találja, hogy TCP, ezért a datagrammot a TCP-nek adja át. A TCP a Sorszám mező tartalma és egyéb információk alapján állítja össze az eredeti állományt.

6.5. Egyéb fontos protokollok

–UDP: A User Datagram Protocol (UDP) az internet egyik alapprotokollja. Feladata datagram alapú szolgáltatás biztosítása, azaz rövid, gyors üzenetek küldése. Jellemzően akkor használják, amikor a gyorsaság fontosabb a megbízhatóságnál, mert az UDP nem garantálja a csomag megérkezését. Ilyen szolgáltatások például a DNS, a valós idejű multimédia átvitelek, vagy a hálózati játékok.
–NTP: Network Time Protokol. Az idő hálózaton keresztüli szinkronizálását végző protokol.
–FTP: File Transfer Protocol. Fájlok le illetve feltöltésére használt protokol.
–POP3: Post Office Protokol. Levelezésnél levelek olvasásához használható protokol, a levelet letölti a levelező kliens a szerverről a helyi meghajtóra.
–IMAP: Internet Message Access Protocol. A POP3 rokona, a leveleink a levelező szerveren maradnak. Sok levelezéshez kapcsolódó szolgáltatást nyújt.
–HTTP: Hyper Text Transfer Protokol. A HTTP egy kérés-válasz alapú protokoll kliens és szerver között.