Kedvenc Kísérleteim

Informatika

Járai Antal

# Kedvenc Kísérleteim

# Informatika

# TARTALOMJEGYZÉK

# BEVEZETÉS

## Az informatikáról

Ez a könyv elsősorban a tizenéves korosztálynak szól. Különösen azoknak, akik közelebbről szeretnének megismerkedni az informatikával, a számítógépekkel. Ha leülünk egy számítógéphez — legyen az óriási vagy csak egy mobiltelefon, esetleg egy hitelkártya méretű vagy még kisebb gép — szinte biztos, hogy egy GUI-vel (Graphic User Interface = grafikus felhasználói felület) találjuk szembe magunkat. Ebből kiválaszthatjuk a használni kívánt alkalmazást. Jól be is gyakorolhatjuk a használatát. A szülők rendszerint nagyon elégedettek, hogy „a gyerek mennyire ért a számítógéphez". Mi kell még? Semmi, egészen addig, amíg a grafikus felület mindenre ad megoldást, amit akarunk. Ha nem, vagy ha magunk akarunk alkalmazást írni, vagy egyszerűen csak programozni akarunk, esetleg mélyebben érdekel, hogy hogyan működik a dolog, akkor ez a könyv nekünk szól.

Egy személyes élmény: nemrég vettem egy kis „EasyCAP" nevű kütyüt, hogy néhány régi VHS kazettát digitalizáljak. Persze, semmilyen értelmes leírást nem kaptam vele, és a kipróbált grafikus felülettel dolgozó szoftverek se tudták használni (sokféle EasyCAP van, de mindet ugyanúgy hívják). A hálózaton utánanézve a dolognak, találtam néhány hasznos és sok haszontalan információt. A leghasznosabb egy informatikusé volt, aki leírta, hogy egy — kicsit nehezebb — problémával hogyan küzdött meg, és hogyan oldotta meg végül az `ffmpeg` parancs segítségével a dolgot. Egyébként a GUI szoftverek közül is sok ezt a parancsot használja, csak nem mindig használják a megfelelő paraméterezést. Végül a megfelelő paraméterezéssel nekem is működött.

Ma a két legelterjedtebb úgynevezett operációs rendszer, amivel találkozunk a Windows és a UNIX különböző változatai. A Windows-ért és a Windows-os szoftverekért (általában) fizetni kell, a UNIX-alapú rendszerek és szoftvereik (általában) ingyenesek, a profik írják őket kedvtelésből. UNIX-változat a Mac OS, a Sun Solaris, az IBM AIX, a MINIX, és a Linux változatok is mind. Az informatikusok többsége a UNIX-alapú rendszereket részesíti előnyben, mert nagyon ritkán kell újraindítani őket (a munkahelyi gépemen leállás nélkül évekig futott egy Open BSD UNIX), és kéretlenül nem töltenek

le semmit és nem indulnak újra. Ráadásul a Windows inkább a GUI szoftvereket támogatja, a UNIX eredetileg parancs módban volt használható, de ma már nagyon jó grafikus felület van hozzá. Ma már minden gépre, még a hitelkártya méretű Raspberry Pi gépekre (kevesebb, mint 30000 Ft) is feltehető valamilyen UNIX változat. Én itt mindent Ubuntu 20.04.2 LTS Linux alatt írtam illetve próbáltam ki, de minden nagyon hasonló más UNIX változatok alatt is. Meg kell kérnetek egy helyi „varázslót", hogy tegyen fel a gépetekre egy UNIX-változatot.

Az informatikát nagyon nehéz úgy megtanulni, hogy nincs előttünk egy számítógép, amin mindjárt ki is próbálhatjuk, amit olvasunk. Tehát *kísérlet*ezni kell. Néha az adott rendszeren valami másként működik. Nyugodtan menjünk tovább. Fokozatosan meg fogjuk tanulni, hogyan deríthetjük ki az eltéréseket. Közben sok adatot is megjegyzünk. *Az elsődleges cél a megértés.* Legjobb a kísérleteket sorban elvégezni. A könyv első olvasásra kihagyható, nehezebb részei $*$ és * jelek között vannak, a még nehezebbek $**$ és ** között. Legjobb a fejezeteket sorban elolvasni, egymásra épülnek. Az első négy fejezet könnyebb, ezek bevezető jellegűek. Az utolsó három fejezet anyaga lényegesen nehezebb, ezek megegyeznek egy, az ELTE Informatika Karán tartott 4 féléves speciálkollégiumom anyagával.

Ismeretterjesztő könyveknél szokatlanul a könyvhöz van mutató. Itt betűrendbe felsorolom a fogalmakat, neveket az oldalszámokkal, ahol előfordulnak. A legfontosabb oldalszám, ahol elmagyarázom a dolgot, *dőlt* betűvel van megadva. Így felesleges mindent többször elismételni. Ha valami olyannal találkozunk, amit kihagytunk, vagy már elfelejtettünk, könnyen megtalálhatjuk a magyarázatot. Ugyancsak megadom a könyv végén, hogy milyen könyveket használtam fel, még akkor is, ha nem magyar nyelvűek. A pontos hivatkozást azonban, hogy mit honnan vettem, nem adom meg, ez ismeretterjesztő könyveknél nem is szokás.

Ez a könyvsorozat öt könyvből áll. Ez a könyv kapcsolódik ugyan a matematikáról, fizikáról és az elektronikáról szóló könyvekhez, de csak gyengén, ömagában is érthető.

Budapest, 2021. február 25. *Járai Antal*

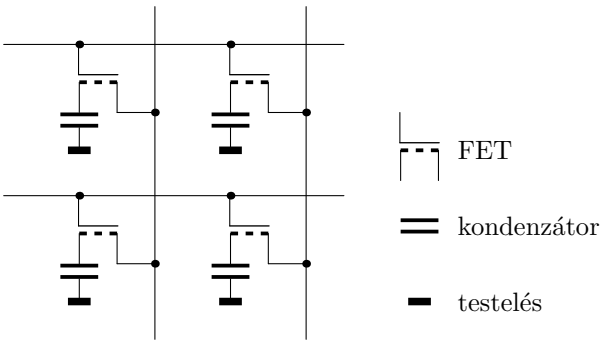
# 1. AZ OPERÁCIÓS RENDSZER

Ha megnézünk egy számítógépet, fizikailag jelenlévően, tapinthatóan a *hardver*t látjuk (*hardware* = vasárú). A gép működéséhez azonban nélkülözhetetlen a működtető program, a *szoftver* (*software* = „lágy árú"). Az a rész, amivel közvetlenül „beszélgetünk" az *operációs rendszer*, pontosabban egy része, a *kernel* (kernel = mag). Ha parancs módban vagyunk — a régi gépeken csak ez volt — parancsokat adhatunk ki, amiket a gép végrehajt. Ha egy szoftver grafikus felülete van előttünk, akkor az egérkattintgatással, menükből való kiválasztással tulajdonképpen szintén parancsokat adunk ki. Itt tulajdonképpen nincs is mit megérteni, legfeljebb megjegyezni. (Néhány grafikus szoftverhez nincs is használati utasítás.) Ha nem akarunk megismerkedni a szoftver (és a gép) „lelkével" illetve a programkészítéssel, akkor nyilván ezt választjuk, még akkor is, ha paracs módban dolgozni sokszor gyorsabb. A mi célunk azonban több.

Az operációs rendszerek közül a *UNIX* valamelyik változatát fogjuk használni. Történetileg a UNIX elődje a *Multics* volt, ami egy nagy és bonyolult operációs rendszer. A cél ennek az egyszerűsítése és jóval kisebbé tétele volt. A munkát a *Bell Labs*-nél nagyrész Ken Thomson és Dennis Richie végezte el. A UNIX Programmer's Manual (= UNIX proramozói kézikönyv) 1971-ben jelent meg. A 4-es változatot széles körben használták a Bell Labs-nél. Ezt 1973-ban újraírták, nagyrészt C nyelven. Ugyanebben az évben tették közzé munkájuk leírását. A 6-os változat 1975-ben jelent meg. Ez maradt a legszélesebb körben használatban az 1980-as évek elejéig, erről szól Lion [4] könyve, amit oktatási célokra írt. Az 1980-as években aztán számos UNIX (vagy Unix) változat jött létre. A *GNU* terv szabadon használható szoftverek létrehozását tűzte ki 1983-ban. Oktatási célokra készült a *MINIX*, szerzője Andrew S. Tannenbaum (1987). Linus Torvalds ennek alapján hozta létre a *Linux*-ot 1991-re, amely a GNU tervhez tartozik 1992-től.

A UNIX kezdettől fogva többfelhasználós volt, azaz többen is használhatják egyszerre ugyanazt a gépet, anélkül, hogy egymást zavarnák. Ezt akármilyen távolról is megtehetik. (A Windows csak 2000-től ilyen.)

Ennek a résznek a célja ismerkedés a UNIX-típusú operációs rendszerekkel. Mielőtt ebbe belekezdenénk, kicsit beszélnünk kell a számítógép felépítéséről.




1.1.1. ábra: DRAM

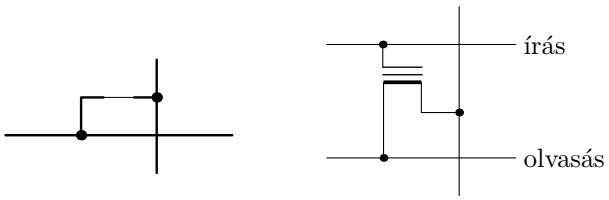
# 1.1. Hardver

**1.1.1. A core.** Bármilyen számítógép kerül a kezünkbe, egészen biztosan van *memóriája*, ahol például a számítások részeredményeit és más adatokat tárolja. Hajdanán a memória búzaszemnyi vagy még kisebb mágneses gyűrűkből állt (az űrrepülőgépben még ilyen volt). Innen ered a néha még használt core (= mag) elnevezés, de ma inkább *központi tár*ról, *operatív tár*ról vagy egyszerűen csak *tár*ról beszélünk. Egy mai elektronikus memória vázlatos kapcsolási rajzát az 1.1.1. ábra mutatja. A jobb oldalon látható a kapcsolási rajz jeleinek a jelentése (erről bővebben a fizika és az elektronika részben). Az ábrán vízszintesen vezetékek futnak, ezek a *sorvezeték*ek, a függőlegesek pedig az *oszlopvezeték*ek. Maga a tároló elem itt egy kis kondenzátor, amely vagy fel van töltve elektromossággal, vagy nincs, tehát két állapota van. Ezt a két állapotot tekinthetjük igaznak illetve hamisnak vagy egynek illetve nullának. Egy-egy kondenzátor állapota tehát két féle lehet, az általa tárolható információmennyiség a lehető legkisebb, egy *bit*. Az ábrázolt részleten négy kis kondenzátor van, tehát négy bitet tárolhat. A valóságban még a legkisebb memória is sokkal több sor- és oszlopvezetéket tartalmaz, például 256-ot mindegyikből, így 65536 bitet tárolhat. Egy ilyen memória tehát 65,5 kb-es. Itt a b a bit rövidítése, a k pedig szokás szerint ezret jelent. Az informatikában rendszerint kettőhatványok szerepelnek, így célszerűbb kettőhatvány, például $2^{10} = 1024$-es egységekben mérni. Ekkor ennek a memóriának a tárolóképessége (kapacitása) 64 kib. A kis i betű azt jelzi, hogy informatikai egységet használunk, nem a szokásosat, tehát ki= $2^{10}$ =1024. Hasonlóan Mi= $2^{20}$, Gi= $2^{30}$, Ti= $2^{40}$ és Pi=$2^{50}$.

Hogyan olvashatók ki a tárolt bitek? Az ábrán jobb oldalt látható egy FET (Field Effect Transistor = térvezérlésű tranzisztor) rajzjele. Három „lába” (kivezetése) van. Az egyik a kondenzátorhoz van kötve, a másik az oszlopvezetékhez. A kettő között szaggatott vonal van, ez a *csatorna*. Az, hogy szaggatott, azt jelzi, hogy alapállapotban a csatorna nem vezet. Megváltozik a helyzet, ha a harmadik kivezetésre a *kapu*ra (gate) pozitív feszültséget (néhány voltot) kapcsolunk. Ez a kapu alá vonzza az elektronokat, és a csatorna vezetni fog. Egyszerre csak egy sorvezetékre kapcsolunk pozitív feszültséget.

Az erre a sorvezetékre kötött kapuk mind pozitív feszültségre kapcsolódnak, és a kis kondenzátorok töltése (ha volt) a megfelelő oszlopvezetékre jut és ott kis áramlökést hoz létre. Az oszlopvezetékekre kötött erősítők érzekelik ezt az áramlökést (mint 1-et) vagy a hiányát (mint 0-t), és ezzel az egész sor tartalmát kiolvastuk. Sajnos, egyúttal a tartalom meg is semmisült, de az eszköz rögtön automatikusan vissza is írja: attól függően, hogy mit olvasott, magas vagy nulla feszültséget ad a megfelelő oszlopvezetékre, és a sorvezeték magas feszültségre emelésével kinyitja a FET-eket. A kondenzátorok feltöltődnek vagy nem. A kondenzátorok másik kivezetése „testelve" van, mind ugyanoda van kötve. A testelés rendszerint a szilícium lapka maga, amin a kondenzátorokat és FET-eket, meg a többi szükséges alkatrészt kialakítják. Az írás teljesen hasonló: kiolvassuk egy sor tartalmát, de az írni kívánt bitekbe nem a kiolvasott tartalmat, hanem a kívülről kapott információt írjuk vissza.

Az egész eszközt — kissé szerencsétlen módon — RAM-nak (Random Access Memory = véletlen hozzáférésű tár) hívják. A „véletlen hozzáférésű" azt akarja jelenteni, hogy a biteket teljesen tetszőleges sorrendben olvashatjuk ki illetve írhatjuk be a tárba. A RAM rövidítésbe mindig beleértjük azt is, hogy a tár írható és olvasható. Tulajdonképpen amit leírtunk az az olcsóbb változat, a DRAM. A D betű a dinamikus rövidítése. A kis kondenzátorok elég rövid idő alatt — nagyságrendileg 0,1 s-tól 10 s-ig — elveszítik a töltésüket. Ennyi idő alatt minden sort ki kell olvasni egyszer, hogy a visszaíráskor „frissüljön". Ez a „frissítés" kezdetben a számítógép többi részének — például a processzornak —a feladata volt, ma már a DRAM maga végzi. Van SRAM (sztatikus RAM) is, még gyorsabb is, de ebben a tároló elem hat FET-et tartalmaz, így drágább. Ezt nem kell frissíteni, de ennek is elvész a tartalma, ha nem kap áramot.




1.1.2. ábra: ROM és EEPROM

**1.1.2. ROM és PROM.** Nyilván nem hasznos, ha gépünk kikapcsoláskor mindent elfelejt. Szükség van olyan memóriára is, amely áram nélkül sem felejt. Az 1.1.2. ábra bal oldalán egy ilyen memória egyetlen „celláját" mutatjuk be, bár a teljes memóriában itt is számos sor- és oszlopvezeték van. A cella nagyon egyszerű, úgy működik, mint egy — gépkocsikban és elektronikus készülékekben is használt — biztosíték. A biztosíték egy üvegcsőben lévő nagyon vékony és könnyen olvadó fémszál. Ha nagy áram megy át rajta, akkor elolvad, és többé már nem vezet. Az ábrán még érintetlen állapotban ábrázoltuk a kis biztosítékot, a vékony szálat vékony vonallal rajzolva. Ebben az állapotban a cella 1-et tartalmaz, ha azonban a szálat megolvasztjuk, akkor megszakad, és 0-t. Egy sor kiolvasása úgy történik, hogy egy sorvezetékre feszültséget — akár csak 1 voltot — adunk.

Ha a szál nincs megszakadva, akkor a feszültség megjelenik az oszlopvezetéken is, ha viszont meg van szakadva, akkor nem. Lehet, hogy a szakadásokat már a gyártó elkészíti (pontosabban egyes cellákba nem is készít vezetéket). Az ilyen eszköz neve ROM (Read Only Memory = csak olvasható memória). Ez csak akkor gazdaságos, ha rengeteg ROM készül ugyanazzal a tartalommal. Sokkal gyakoribb az úgynevezett PROM: a P betű azt jelenti hogy programozható. A gyárból úgy kerül ki, hogy minden kis biztosíték sértetlen, azaz az egész memória 1-ekkel van teleírva. A számítógép gyártója „programozza" ezt a memóriát, azaz írja be a nullát, ahova kell. Ez soronként történik: egy sorvezetékre magas (> 10 V) feszültséget kapcsolunk. Azokra az oszlopvezetékekre, amelyekhez tartozó cellában meg kell maradni az 1-nek, ugyanezt a feszültséget kapcsoljuk, és nem történik semmi, mert nem folyik áram. Ahova 0-t akarunk írni, azt az oszlopvezetéket 0 feszültségre kapcsoljuk, és a nagy áram hatására a kis biztosíték kiolvad, többé már nem vezet. A kiolvasás és az írás is bitenként történhet, azaz ez a fajta memória is véletlen hozzáférésű.

**1.1.3. EEPROM, flash és Winchester.** Természetesen a PROM nem az, amit akartunk: csak egyszer lehet írni. Szükségünk van valamilyen írható-olvasható és nem felejtő *háttértár*ra, *tömegtároló eszköz*re. A PROM-hoz hasonló, de többször írható az EEPROM (Electrically Erasable PROM). Egy cellát mutat az 1.1.2. ábra jobb oldala. A cella tulajdonképpen egy speciális FET tranzisztor, amelyben a kapu alatt, a csatornától nagyon vékony, de igen jól szigetelő réteggel elválasztva egy „lebegő kapu" elektróda is van. Alapállapotban a lebegő kapun nincsenek többlet elektronok, és a csatorna vezet: ez van a rajzon ábrázolva. Ha valahogy többlet elektronokat viszünk a lebegő kapura, akkor negatívvá válik, eltaszítja a vezetést adó elektronokat a csatornából és a csatorna már nem vezet, „szaggatottá válik". Az egyszer felprogramozott EEPROM ugyan úgy olvasható alacsony feszültséggel, mint egy PROM. Hogyan lehet bele írni? Az előre beírt egyesek közül kell némelyiket nullára változtatni. Adjunk a sorhoz tartozó írás vezetékre magas (> 10 V) feszültséget, míg a megfelelő oszlopvezetékre nullát, ha írni akarjuk a cellát! Az elektronok az oszlopvezeték felől alagúthatással (erről lásd részletesebben a fizika részt) feljutnak a lebegő kapu elektródára, és ott nagyon tartósan (akár 100 évig is) megmaradnak. Persze, ahova nem akarunk nullát írni, ott az oszlopvezetéket ugyanakkora feszültségre kapcsoljuk, mint az írás vezetéket. A törlés csak soronként lehetséges. A sorhoz tartozó olvasás vezetékre nagy feszültséget, az írás vezetékre pedig nullát kapcsolunk. A feltöltött lebegő kapuk töltése az olvasásvezeték felé alagúteffektussal távozik, és az egész sor törlődik, pontosabban egyre íródik át. Más vezetékezés is lehetséges. Az EEPROM és a flash (= villanás) memória között nincs lényeges elvi különbség. Ha a törlés kisebb egységekben és sokszor (nagyságrendben milliószor) lehetséges mielőtt az eszköz tönkremegy, inkább EEPROM-ról beszélünk. Ha a törlés nagy egységekben (akár az egész memória egyben) és kevesebbszer lehetséges, akkor flash memóriáról beszélünk. Persze a flash memória cserébe jóval nagyobb kapacitású. Ilyen memória van például a merevlemez nélküli gépekben, a pendrive-okban, a fényképezőgépek memória kártyáiban. Az EEPROM az olvasás és az írás szempontjából véletlen hozzáférésű, de a törlés szemponjából csak blokkonként férhető hozzá.

Másik elterjedt tömegtároló és nem felejtő eszköz a *Winchester*, *mágneslemez* vagy

*merev lemez*. Nevét az angliai Winchester városka nevéből kapta, az IBM ottani gyárában fejlesztették ki. Kezdetben kapacitása néhányszor 10 Mb volt, ma néhányszor 10 Tb. Ma még olcsóbb és tovább használható, mint a flash memória, de rázásra érzékenyebb. Nagyjából azon az elven működik, mint a magnetofon. Egy vagy több tükörsima alumínium disc (= korong) egyik vagy mindkét oldala speciális kobalt ötvözettel van bevonva. A korong 10000 fordulat/perc nagyságrendű sebességgel forog. Rugók szorítanak a felületéhez író illetve olvasó fejeket. A fejek „szárnyakkal" vannak ellátva, és szó szerint repülnek a mágneses réteg felett 1 nm nagyságrendű távolságban. Az író fej megmágnesezi az alatta lévő mágneses réteg egy kis tartományát, rendszerint a rétegre merőleges irányban. Hogy az északi sarok kifelé vagy befelé irányul, az az író fej tekercsében folyó áram irányától függ. Az egyik nullát, a másik egyet jelent. Az olvasó fej érzékeli a mágnesezés irányát. Az elvekről részletesebben lásd a könyvsorozat fizika részét. Az olvasásra (és az írásra is) 10 ms nagyságrendű időt várni kell, míg a fej ki-be mozogva megtalálja a szükséges sávot és a keresett szektor a fej alá érkezik, de azután már nagy sebességű, ha egymás utáni szektorokat olvasunk vagy írunk. Régebben hajlékony mágneslemezeket, még régebben pedig mágnesszalagot is használtak tömegtárolásra, ez mára kiszorult a használatból, ugyanúgy, mint a nyolc csatornás (papír) lyukszalag illetve a nyolcvan oszlopos IBM (papír) lyukkártya.

**1.1.4. A processzor.** A szó jelentése *feldolgozó egység*: ez végzi a számítógépben a műveleteket bitsorozatokon. A végezhető műveletek összessége a processzor *utasításkészlete*. A processzor egyik jellemzője a *szóhossz*; milyen hosszú bitsorozattal tud egyszerre dolgozni. Ez kezdetben tetszőleges volt, lehetett például 24 vagy 36 bit. Az első *mikroprocesszor* (egyetlen integrált, azaz összetett áramkörből álló *processzor*), az Intel 4004 processzora 4 bites volt és 1971-ben jelent meg. Egy évvel később követte a már 8 bites 8008, majd 1974-ben a 8080. A Zilog cég 1975-ben dobta piacra a Z80 processzort, amely a 8080 továbbfejlesztése. Mindezeket ugyanaz a mérnök tervezte. Közben több más gyártó is jelentkezett 8 bites mikroprocesszorokkal.

Az első 16 bites mikroprocesszor a Texas Instruments TMS9900 processzora volt 1976-ból. Az Intel első 16 bites processzora 1978-ban jelent meg. Ez és továbbfejlesztett változatai aztán nagyon elterjedtek x86 család néven, mivel — elég érthetetlen módon — az IBM ennek egy gyengített változatát, a 8088-at választotta az 1981-ben megjelent IBM PC (Personal Computer - személyi számítógép) processzorának, pedig ekkor már a mérnököknek szánt gépekben mind 32 bites processzorok, elsősorban Motorola 68000-esek dolgoztak. (Maga az IBM is ezt használta nagy gépeinek helyettesítésére tervezett nagy teljesítményű PC-iben.) A Motorola 68000-es (röviden 68k) sorozatának első processzora 1979-ben jelent meg.

1960-ig a gépek utasításkészlete meglehetősen egyszerű volt, de ezután kezdett egyre bonyolultabbá válni. Az eddig említett mikroprocesszorok mind CISC típusúak (Complex Instruction Set Computer = bonyolult utasításkészletű számítógép). AZ IBM-nél 1975-től végzett elemzések megmutatták, hogy sokkal egyszerűbb utasításkészlet is elegendő és a processzor gyorsabb lesz: ez a RISC-elv (Reduced Instruction Set Computer = redukált utasításkészletű processzor). Az elv alapján egyetemek is terveztek és gyártottak processzorokat. 1992-ben indult a RISC-elvű DEC Alpha 64 bites processzor gyártása,

amely vagy háromszor gyorsabb volt riválisainál, de az Intel felvásárolta és megszüntette a gyártást, hátráltatva ezzel a fejlődést. Igazán a POWER mikroprocesszor családdal lépett szinpadra az elv (IBM-Motorola-Apple, 1993). Ezek máig is a leggyorsabb processzorok, főleg szerverekben használatosak. A világon legtöbbet gyártott ARM (Advanced RISC Mashine = továbbfejlesztett RISC gép) első, ARM1 (32 bites) változata kevesebb, mint feleannyi alkatrészt tartalmaz, mint a 68k, és sokkal kevesebb áramot fogyaszt. Az ARM különböző 32 és 64 bites változatai ma a világ legelterjedtebb mikroprocesszorai, ott vannak az összes „okos" eszközben.

A processzor másik fontos jellemzője az órajel sebsssége. Hz-ben mérjük, ami tulajdonképpen 1/s; nagyjából azt mondja meg, hány utasítást tud a processzor végrehajtani másodpercenként. Nem pontosan, mert a régebbi mikroprocesszoroknál egy utasítás végrehajtása több órajelig tartott, a mai processzorok viszont több utasítást is el tudnak indítani egy órajelben. Ráadásul ma egy tokba több processzor „magot" is raknak, és még más trükkök is vannak. A Z80 maximális órajele 4 MHz volt, ma a maximális órajel 4 GHz körül jár.

**1.1.5. Be- és kiviteli eszközök.** A beviteli eszköz alapvetően a billentyűzet, kiegészítve esetleg az egérrel vagy érintőképernyővel. Működni ugyanúgy működik, mint a ROM. Sor- és oszlopvezetékek vannak benne, ha nem is egyenesek. A sorvezetékekre sorban feszültséget adva, és megvizsgálva, hogy egy oszlopvezetéken van-e feszültség, eldönthető, hogy egy adott billentyű le van-e nyomva.

A kiviteli eszköz alapvetően a képernyő. Az elv hasonló. Minden képponthoz (pixel) három világító dióda tartozik: egy piros, egy zöld és egy kék. Minden színhez vannak oszlopvezetékek. Egy adott sorvezetékre feszültséget adva, az adott szín oszlopvezetékével szabályozhatjuk, hogy a sor egy pixeléhez milyen erős piros, zöld és kék szín tartozzon. Ez megadja az adott sor pixeleinek fényerejét és színét. A sor világító diódái nagyon rövid ideig villannak fel, utána másik sor következik. Másodpercenként legalább 25-ször minden sor sorrakerül. Persze, vannak más elvű kijelzők is.

**1.1.6. Perifériák.** A külső eszközök vagy *perifériák* ma szinte kivétel nélkül USB (Universal Serial Bus = univerzális soros gyüjtősín) segítségével csatlakoztathatók a számítógépre. (A párhuzamos bus-ok mára a számítógép belsejébe szorultak vissza.) A perifériák a legkülönbözőbbek lehetnek, például külső merevlemezek, játékkonzolok, stb. Az USB 1.1 sebessége 12 Mb/s és kétféle, a közönséges A és a ritkább B típusú csatlakozó volt hozzá. A ma legelterjedtebb USB 2.0 sebessége 480 Mb/s és az előző két csatlakozó mellett mindkettő mini és mikro változatban is létezik hozzá. Az USB 3.2 Gen 1 sebessége 5 Gb/s, míg az USB 4-é 20 Gb/s. Ez utóbbinál már csak USB C típusú csatlakozó használható. Kényelmes, hogy az operációs rendszer az USB-n csatlakoztatott eszközöket felismeri. A megengedett kábelhossz az USB 4-nél már csak 0,8 m. Ekkora távolságot az elektromos jel — amely majdnem fénysebességgel halad — nagyjából 4 ns alatt tesz meg. Istent nem lehet megvesztegetni: a fénysebesség és a hossz meghatározza a késleltetést, ez pedig korlátozza az átviteli sebességet. A gép belsejében drága párhuzamos bus-okon (akár 256 bit párhuzamosan) nagyobb átvitt bitmennyiséget lehet elérni, de ott is csak kis távolságra, egyébként zavar a késleltetés.

Másik fontos csatlakozó (= port) az Ethernet csatlakozó, ma már 1 Gb/s átviteli

sebességgel, akár 100 m-re is. Van még egy „láthatatlan" csatlakozás, a Wi-Fi. Ez rádióhullámokkal nagyon kis hullámhosszal működik. A falak általában zavarják, csak addig biztos, amíg az adó és a vevő látják egymást. Persze, a távolsággal is gyengülnek a jelek. A fizikai alapokat lásd a fizika részben.

**1.1.7. Egy példa: a ZX81.** Az első személyi számítógép, amely 100 $ alatti áron volt kapható, a ZX81 volt (Sinclair, 1981). Processzora 3,25 MHz-es órajellel működtetett Z80 volt. Tartalmazott egy 8× 8 kib-es ROM-ot és egy 8×2 kib-es RAM-ot, ami 8×16 kib-tel bővíthető volt. Az összes többi elektronikai alkatrészt egyetlen integrált áramkör tartalmazta. A képernyő tartalma és más adatok 8×1 kib-et foglaltak le, így alapkiépítésben a programnak 8×1 kib maradt. A bevitelre *vezető gumi*ból készült billentyűzet szolgált. Ez gumiba kevert fémreszelékből áll. Alapállapotban nem vezet, de ha összenyomjuk, a fémrészek összeérnek és a gumi vezet. Egy ilyen gumilapnak a hátulján voltak a sor- és oszlopvezetékek, és az elejére voltak nyomtatva a betűk és más jelek. Ez volt felragasztva a gép dobozának tetejére. Képernyő nem volt, hanem a gép egy TV-jelet hozott létre, amit egy TV-be lehetett vezetni. Lassú üzemmódban a processzor idejének 3/4 részét a TV-jel létrehozásával töltötte. Gyors üzemmódban nem volt kép. BASIC (Beginners' All-purpose Symbolic Instruction Code = kezdők általános célú szimbolikus utasítás kódja) nyelven volt progamozható, amelynek értelmezője minden mással a ROM-ban volt. Háttértárként kazettás magnetofont lehetett használni a fülhallgató kimenetel és a mikrofon bemenettel.

## 1.2. Néhány parancs

**1.2.1. Kísérlet.** Aki feltette a gépünkre a Linux-ot vagy más UNIX-változatot, attól megkapjuk a login nevünket (login = bejelentkezés) és a *jelszó*nkat. Ha van **login:** felirat, akkor oda gépeljük be a login nevünket, a **password:** (= jelszó) felirathoz pedig a jelszónkat. Ezzel léphetünk be a gépre. Valószínűleg egy grafikus felülettel találjuk szemben magunkat. Keressünk egy fekete ikont, amelyben fehér betűkkel >_ áll, vagy valami hasonló. Arra rákattintva egy úgynevezett „prompt"-ot (prompt = sarkall) kapunk, ami rendszerint egy `$` jel, és jelzi, hogy a gép várja a prancsunkat.

**1.2.2. Kísérlet.** Kezdjük a legegyszerűbb paranccsal, az `echo` (visszhang) parancscsal. Gépeljük be, hogy

```
$ echo alfa
```

A válasz:

```
alfa
```

Ha azt gépeljük be, hogy

```
$ echo alfa beta
```

a válasz

```
alfa beta
```

Ha most azzal próbálkozunk, hogy

```
$ echo alfa   beta
```

a nem várt válasz

```
alfa beta
```

A szóközökből csak egy maradt. Ezt nem az echo parancs csinálta, hanem az `sh` (shell = burok) parancsértelmező. Amit begépeltünk — miután befejeztük az esetleges jobbrabalra nyilakkal mászkálást, törléseket (delete = törölj), beszúrásokat (insert =szúrj be), visszatörléseket (backspace = visszaszóköz) és egyéb javítgatásokat, és leütöttük az *Enter* billentyűt — mindent a burok vesz munkába: szavakra bontja, különböző speciális jelentésű karaktereket vesz figyelembe, majd megkeresi az első szó által megadott parancsot, és annak adja át a vezérlést. Elsősorban a saját belső parancsai között keres — az `echo` parancsot már itt megtalálja —, aztán az operációs rendszer többi parancsa között, végül a gépen lévő parancsok, programok között. Ez azt jelenti, hogy akárki írhat újabb parancsokat! Itt most a burok ténykedése írtott ki két szóköz karaktert, amikor szavakra bontotta a bemenetet. A bemenetet vagy részeit megvédhetjük a buroktól, ha idézőjelbe tesszük:

```
$ echo "alfa   beta"
alfa   beta
```

Ugyanígy használható a ' félidézőjel is. A félidézőjelek közé tehetünk idézőjeleket és fordítva is. Most próbáljuk ki az

```
$ echo alfa # beta
```

sort. A válasz

```
alfa
```

Mi történt itt? Megint a burok ügyködött. Számára a `#` speciális karakter, ami azt jelenti, hogy a sor további része megjegyzés, kommentár (comment = megjegyzés), nem kell vele semmit sem csinálni. Idézőjellel persze védhetjük a `#` jelet, de van egy másik módszer is. Írjunk a `#` elé egy \ jelet! A \ (backslash = fordított törtvonal) megvédi az utána következő speciális karaktert attól, hogy a burok értelmezze, és változatlan formában kerül az `echo`hoz:

```
$ echo alfa \# beta
alfa # beta
```

Próbáljuk ki, hogy

```
$ echo alfa# beta
alfa# beta
```

A `#` csak akkor viselkedik speciális karakterként, ha szó elején van. Persze, a \ is speciális karakter, így azt is védeni kell a buroktól idézőjellel vagy egy másik \ karakterrel, ha azt akarjuk, hogy változatlanul kerüljön az `echo` parancshoz:

```
$ echo alfa \\ beta
alfa \ beta
```

Más, a burok számára speciális karaktert is így védhetünk meg a buroktól. Egy kivétel van: az *Enter* elé \ jelet írva, törli az *Enter* karaktert. Ugyanez a hatás idézőjelek között

is. Ez nagyon hasznos, ha hosszú sort akarunk begépelni, jobban érthető lesz a bemenet. Itt egy példa ennek a lehetőségnek a használatára:

```
$ echo abc\
> def\
> ghi
abcdefghi
```

Figyeljük meg, hogy a második és a harmadik sorban megváltozott a prompt: `$` helyett `>` áll, jelezve, hogy a sor még nincs befejezve.

A `#` megjegyzés jel és az `echo` parancs is tökéletesen értelmetlennek tűnik egyelőre. A burok azonban programozható. Ha egy sok sorból álló burok programot készítünk, nagyon is célszerű, ha megjegyzéseket helyezünk el benne. Ezek később nagyon sokat segítenek, ha javítanunk kell benne. Az `echo` parancsokkal pedig egy sok sorból álló burok programban tájékoztathatjuk a program felhasználóját, hogy hol is tart, mit csinál a program. A burok programozásával később foglalkozunk, hiszen még a programozás alapjaival sem ismerkedtünk meg.

**1.2.3. Kísérlet.** Tudnánk-e már most valami hasznosat csinálni az `echo` paranccsal? Igen, hála a UNIX rengeteg lehetőségének. Sok parancs a szabványos kimenetre (standard output = szabványos kiment) ír, az `echo` is. Ezt azonban meg lehet változtatni: írhatunk egy tetszőleges fájlba, azaz állományba is (file = állomány). Tegyünk a szöveg után egy > jelet, és utána a fájlnevet:

```
$ echo "Lenni vagy nem lenni;" >teszt1.txt
```

Kimenet nincs az *Enter* után, az `echo` eredménye a `teszt1.txt` nevű állományba ment. Vigyázat, ha volt `teszt1.txt` nevű állomány, annak a tartalma VÉGLEG törlődött. A UNIX feltételezi, hogy nem vagyunk kétbalkezesek, és tudjuk, mit csinálunk. Ha nem vagyunk biztosak benne, hogy nincs ilyen fájl, előbb adjuk ki a

```
$ ls teszt1.txt
```

(list = listázz) parancsot! Ha van ilyen fájl, visszakapjuk a nevét, ha nincs, hibajelzést kapunk.

Szeretnénk megnézni az új fájl tartalmát. Ezt megtehetjük többek között a `cat` (concatenate = vond össze) paranccsal:

```
$ cat teszt1
Lenni vagy nem lenni;
```

A `cat` parancs arra készült, hogy több fájl tartalmát összevonja. Hozzunk létre egy másik fájlt! Ezt megtehetjük a `cat` paranccsal is, a következőképpen:

```
$ cat >teszt2.txt
ez itt a kérdés.
```

Ha nincsenek ékezetes betűk a billentyűzeten, írjunk ékezet nélkülieket! A `cat` — ha nem adtunk meg fájlokat — a *szabványos bemenet*ről — ami általában a billentyűzet — várja a bemenetet és most a második sort is mi gépeltük be. Hiába ütjük le azonban az *Enter* billentyűt, a promt nem jön vissza! Írjunk be egy *Ctrl*-*d*-t, azaz tartsuk lenyomva a *Ctrl* billentyűt, és írjunk egy `d` betűt! Ez a bemenet végét jelzi, és a fájl lezárul, a prompt

visszajön. Egyébként a buroknak is ez jelzi a bemenet végét, erre a terminál bezárul. Van a bemenet szabvány bemenetre irányításának egy másik módja is, a << jel utána egy karaktersorozattal. Például <<EOF a szabvány bemenetről veszi a karaktereket amíg egy EOF betűsorozat nem érkezik. Nem szoktam használni. Most kipróbálhatjuk a cat eredeti funkcióját:

```
$ cat teszt1.txt teszt2.txt
Lenni vagy nem lenni;
ez itt a kérdés.
```

Természetesen a cat kimenetét átirányíthatjuk egy mondjuk teszt.txt nevű fájlba, így létrehozva az össszevont fájlt. Tegyük is meg! A cat paranccsal listázhatjuk az eredményt.

**1.2.4. Kísérlet.** Ha elgondolkozunk, feltünhet, hogy a kimenetet két sorba írta ki a cat. Honnan tudta, hogy hol kezdődik az új sor? Tulajdonképpen mi van a fájlban? Ezt pontosan megnézhetjük az od (octal = nyolcas; dump = kitaccsol) paranccsal:

```
$ od teszt.txt
```

Az eredmény elég értelmetlennek látszik: minden sor elején egy hétszámjegyű szám áll, utána pedig 8 darab 6 számjegyű. A sor elején álló számok sorszámnak látszanak, az utána lévők pedig mind 0-val vagy 1-gyel kezdődnek. A program nevéből arra következtethetünk, hogy minden nyolcas számrendszerben van. Mivel a sorszám soronként „20"-szal növekszik, amely nyolcas számrendszerbeli szám 16 a tizes számrendszerben, a kinyomtatott hatjegyű számok valami „dupla" valamit jelentenek. Mivel úgy tűnik, az első számjegy csak 0 vagy 1 lehet, a „dupla" valami 16 bit, tehát a „szimpla" valami 8 bit. Ez a *byte* vagy *bájt*. Szokásos rövidítése B, például 1 kiB = 1024 byte. Nagyjából 1960 óta a számítógépek általában byte szervezésűek, ami azt jelenti, hogy a memória legkisebb elérhető egysége a byte. A bájt felének is van neve, *nybble*. Két bájtra a *wyde* elnevezést, négyre a *tetrabájt* vagy *tetra*, nyolcra az *octabájt* vagy *octa* elnevezést fogom használni. Azokon a processzorokon, amelyeknek az „őse" 16 bites (például az Intel processzorok), a wyde-ot word-nek (= szó) szokták nevezni, és duplaszóról, tetraszóról, stb., beszélnek. Azokon a processzorokon, amelyeknek az „őse" 32 bites, szó alatt tetrabájtot értenek, és beszélnek félszóról, stb.

Térjünk vissza az od parancshoz! Ha bájtonként akarjuk látni a szöveget, adjuk ki az

```
$ od -b teszt.txt
```

parancsot. (Felfelé és lefelé nyilakkal keresgélhetünk az előző parancsok között, és át is szerkeszthetjük őket.) A -b *opció* azt mondja az od parancsnak, hogy bájtonkénti kimenetet produkáljon. Nyolcas számrendszerben megadott számokat látunk, amelyek nagy része 0-val vagy 1-el kezdődik. Ez valamiféle kód, minden bájt lényegében egy betűt jelent. A ma leginkább használt kódok őse az ASCII kód (American Standard Code for Information Interchange). Ez tulajdonképpen hét bites kód, a legfelső bit nulla vagy más célra használják.

Ha jobban szeretnénk magukat a betűket látni, írjuk be az

```
$ od -a teszt.txt
```

parancsot! Most megoldódott az új sor rejtélye! Az sp nyilván a szóköz (space) rövidítése, az `nl` pedig newline (új sor) karakteré. Az `echo` parancs tehát egy új sor karaktert ír az után, amit visszhangoz. Ezt meg lehet szüntetni egy opcióval:

```
$ echo -n alfa
alfa$
```

Észrevehetjük hogy az `od -a` levágja a kódok felső bitjét, mert közönséges nyomtatható karaktereket ír ki. Az

```
$ od -c teszt.txt
```

eredménye már jobban hasonlít arra, amit szeretnénk. A nyomtatható karaktereket kinyomtatja, az új sor \n, ha a felső bit 1 akkor kiírja a kódot oktálisan. A nyolcas számrendszer előnye az, hogy a számok számnak látszanak, de — hacsak a szóhossz nem osztható hárommal — más előnye nincs. A bájt szervezéshez sokkal jobban illik a *hexadecimális* (röviden *hex*), azaz tizenhatos számrendszer. Ebben a,b,c,d,e,f (illetve nagybetű megfelelőjük) is számjegy, rendre decimális 10, 11, 12, 13, 14 és 15. Ha kipróbáljuk a

```
$ od -x teszt.txt
```

parancsot, nem a várt eredményt kapjuk: a szöveget wyde-onként kapjuk, méghozzá (valószínűleg) a nagyobb sorszámú bájt lesz elől. A mi számunkra a

```
$ od -t x1z teszt.txt
```

parancs lesz a megfelelő, próbáljuk ki!

Szövegben a hex számokat `0x`-szel szoktuk kezdeni: a 0 jelzi, hogy szám, az `x` pedig, hogy hex. Látjuk, hogy a szóköz `0x20`, az új sor `0x0a`. Az ASCII kódtábla első 32 helyén különböző vezérlő karakterek állnak, kezdve a `0x00` NUL karakterrel. Eggyel már találkoztunk is, a `0x0d` karakterrel, amelynek eredeti jelentése kocsi-vissza (az írógépen): *Ctrl-d* formában gépeltünk be a szöveg végének jelzésére. Egy másik gyakran használt karakter a `0x0c`, amit *Ctrl-c*-ként gépelhetünk be, és futó programok megállítására használható (eredeti jelentése lapdobás). A kódtábla utolsó karaktere sem nyomtatható, a `0x7f` a *Del* törlés (delete = törölj).

**1.2.5. Kísérlet.** Mire az Olvasó idáig ér, biztosan úgy érzi, ezt nem lehet mind megjegyezni. Nem is kell! Nagyon sok információ lekérdezhető a `man` (manual = kézikönyv) parancs segítségével. Például a

```
$ man ascii
```

parancs kilistázza az ASCII kóddal kapcsolatos tudnivalókat tartalmazó man page-et (page = lap). Itt utalás van a `charsets` lapra, ahol további dolgokat tudhatunk meg a karakter kódokról, például azt, hogy az ISO 8859 (International Standard Organization = nemzetközi szabványügyi szervezet) szabványban az ASCII kódtáblát ékezetes és más betűkkel egészítették ki. A magyar nyelv betűi az ISO-8859-2 kódtáblában vannak. A mai UNIX változatok egyébként a világ összes jelének kódolására alkalmas Unicode (ISO 10646) egy tömörített formáját, az `utf-8`-at használják. Az Unicode alapvetően16 bites kód, de ez nem elég, ezért vannak benne 31 bites kódszavak is. Mindegyikhez van lap és hivatkozás a `charsets` lapon.

A man page-ek 9 csoportba tartoznak. Néhány szó több különböző csoportban is előfordul, például van `man(1)` és `man(7)`. A parancsok mind az egyes csoportban vannak, mindegyikhez van man page. A csoportot opcióként megadhatjuk a `man` parancsnak, de ez nem fontos. Tréfásan azt mondhatjuk, hogy elég annyit tudni, hogy

```
$ man 1 man
```

Próbáljuk ki a

```
$ man echo
```

és a

```
$ man od
```

parancsokat! Mint látjuk, nem érdemes az opciókat megjegyezni, legfeljebb néhány fontosabbat. Érdemes viszont csinálni egy `unix.txt` fájlt, amibe a fontosabb dolgokat beírjuk. A >> átirányítás nem törli a fájlt, hanem a végéhez hozzáír. Így például

```
$ echo "..." >>unix.txt
```

paranccsal gyarapíthatjuk gyűjteményünket.

**1.2.6. Kísérlet.** Szeretnénk a „kísérleti" fájljainkat elkülöníteni a többitől. Egy könyvtárba érdemes átrakni öket. Csináljunk egy könyvtárat az `mkdir` (make directory = csinálj könyvtárat) paranccsal, legyen a neve `tesztek`:

```
$ mkdir tesztek
```

A könyvtár neve nem lehet azonos már létező fájl vagy könyvtár nevével. Az `mv` (move = mozgasd) paranccsal mozgathatjuk a fájljainkat:

```
$ mv teszt*.txt tesztek
```

Mi is történik itt? A `*` speciális karakter a burok számára, 0 vagy több karakterből álló akármilyen karaktersorozatot jelent. A burok megkeresi az éppen aktuális könyvtárban azokat a fájlokat, amelyek neve illeszkedik erre a mintára, és azokat teszi a `teszt*.txt` helyére. Most tehát ennek a helyére `teszt1.txt teszt2.txt teszt.txt` kerül. Az utolsó név könyvtárnév kell legyen. Ha csak két név van, akkor az első fájlt átnevezi a másodikká. Vigyázat, ha volt ilyen fájl, TÖRLŐDIK! A `*` mellett speciális karakter a `?` is, ez azonban pontosan egy tetszőleges karakterre illeszkedik. Tehát a

```
$ mv teszt?.txt
```

parancs csak két fájlt mozgatott volna az új könyvtárba. A `mv` parancshoz teljesen hasonlóan működik a `cp` (copy = másolj) parancs.

**1.2.7. Kísérlet.** Hogyan léphetünk be az új könyvtárba? A `cd` (change directory = válts könyvtárat) paranccsal. A könyvtár tartalmát akár az `echo` paranccsal is megnézhetjük:

```
$ echo *
```

A trükk az, hogy a burok kicseréli a `*` karaktert az illeszkedő fájlnevekre. Próbáljuk ki azt is, hogy

```
$ echo teszt?.*
```

Ebben a könyvtárban is létrehozhatunk egy újabb (al)könyvtárat:

```
$ mkdir tesztek
```

Mint látjuk lehet ugyanaz a neve, mint a könyvtárnak, amiben van. Menjünk bele és hozzunk létre ebben a `cat` paranccsal egy `rendetlen.txt` nevű fájlt, számos rövid, elég össze-vissza sorral.

A fájlnevek listázására szolgál az `ls` (list = listázz) parancs. Főbb opciói `-l, -a, -t, -r`, jelentésük sorra long, all, time és reverse. Alapértelmezésben a parancs névsor szerint listáz, de mint látjuk, idő szerint is listázhatunk, és a sorrendet meg is fordíthatjuk. Próbáljuk ki az

```
$ ls -la
```

parancsot! (Mint látjuk, az egy betűs opciókat egybe is írhatjuk.) Soronként listázza a fájlokat. A legelső betű a file típusa (könyvtárnál d, közönséges fájlnál -, de van más lehetőség is), azután 9 betű és egy számjegy, amiket majd megbeszélünk, a tulajdonos neve, a csoport neve, a fájl hossza bájtban, az utolsó módosítás dátuma, és a fájl neve. Nem csak az egyetlen `rendetlen.txt` fájlt listázza azonban az `ls`, hanem egy . és egy .. nevet is. Ezek az aktuális könyvtár, és az a fölött lévő könyvtár. Ezek is használhatók könyvtár névnek. Például adjuk ki a

```
$ cd ..
```

parancsot, és a „felső” `tesztek` könyvtárba kerülünk. Vegyük észre, hogy a `*` helyére nem helyettesítődött be a . és a .. és semmilyen .-tal kezdő filnév sem, mert ez zavart okozna.

**1.2.8. Kísérlet.** Ha elvesztettük a fonalat, hogy hol is vagyunk, adjuk ki a `pwd` (present working directory) parancsot:

```
$ pwd
/home/.../tesztek
```

ahol a `...` helyén a login nevünk áll. A válasz az aktuális könyvtár *teljes fájlneve*. A „legfelső” könyvtár neve /, vagy úgy is tekinthetjük, hogy üres. Nevezik *root* (gyökér) könyvtárnak is. Nézzünk szét benne:

```
$ cd / ; ls -la
```

Amint látjuk több parancsot is ki lehet adni egy sorban, ha ; van köztük. A kapott listában főleg könyvtárak vannak, néhány fontosabb:

/bin: végrehajtható programok („bináris” fájlok);
/boot: innen indul a rendszer;
/dev: eszközmeghajtók;
/etc: egyéb rendszerfájlok;
/home: a felhasználók könyvtárai;
/lib: fontosabb könyvtárak;
/tmp: ideiglenes fájlok, újraindításkor törlődnek;
/usr: felhasználói könyvtárak.

Mint látjuk, mindnek a tulajdonosa a *root*, más néven *superuser* (= *rendszergazda*. A saját (fő)könyvtárunkba a

```
$ cd
```

paranccsal mindig visszatérhetünk. Tegyük is meg! Egyébként ennek a neve helyett a ~ használható. Ha egy fájlt (vagy könyvtárat, ami szintén fájl) el akarunk érni, az elérési útvonalat (path = ösvény) megadhatjuk az aktuális könyvtártól, a főkönyvtárunktól vagy a gyökérkönyvtártól indulva. Az ösvény begépelését megkönnyíti a *Tab* billentyű használata, amely megpróbálja kiegészíteni az eddig begépelt fájlnevet. Ha ez nem egyértelmű, „csenget", még kétszer megnyomva pedig kiírja a választási lehetőségeket.

A főkönyvtárunkban adjuk ki az

```
$ ls -la
```

parancsot! Találunk majd néhány .-tal kezdődő fájlnevet. Ezeket éppen azért nevezték el így, mert általában nem kell, hogy listázódjanak. Egyébként az `ls` egy könyvtárnevet vár, nem kellett volna a gyökérkönyvtárba menni, hogy tartalmát kilistázzuk, egyszerűen kiadhattuk volna az

```
$ ls /
```

parancsot is. Próbáljuk ki! Most is kiadhatjuk az

```
$ ls tesztek
```

vagy még jobb, az

```
$ ls tesztek/
```

parancsot. Ez utóbbi azért jobb, mert egyértelművé teszi, hogy könyvtárra gondolunk. Más parancsoknál (például `mv, cp, ...`) is hasznos ez.

Menjünk a `tesztek` könyvtárba és adjuk ki az

```
$ ls -l t
```

parancsot! Nem csak a `t` betűvel kezdődő fájlokat listázza ki, hanem az így kezdődő könyvtárakat és azok tartalmát is, egy mélységben.

**1.2.9. Kísérlet.** A második mélységben lévő `tesztek` nevű könyvtárat szeretnénk kitörölni. Könyvtárat kitörölni általában a

```
$ rmdir könyvtárak
```

általános alakú paranccsal lehet; itt a *könyvtárak* könyvtárneveknek egy sorozata. Ha kipróbáljuk, hibajelzést kapunk, hogy a könyvtár nem üres. Valóban, ha belemegyünk és kiadunk egy

```
$ ls -l
```

parancsot, látjuk, hogy itt van a `rendetlen.txt` nevű fájl. Fájlt törölni az

```
$ rm fájlok
```

(remove = törölj) paranccsal lehet. Óvatosan bánjunk vele, mert nem kérdez, csak töröl! „Gyengíteni" a `-i` (interactive) opcióval lehet, ezzel mindig rákérdez, mielőtt töröl. A `-r` (recursive) opció viszont mintegy „felerősíti" a hatását: könyvtárakat is töröl, a bennük lévő fájlokkal és könyvtárakkal, akármilyen mélységben. Egyelőre ne próbáljuk ki a parancsot.

Mi van, ha szeretnénk megvédeni egy fájlt a törléstől? A titokzatos kilenc betű, amit az `ls -l` kiír, itt jut szerephez. A betűk sorban `rwxrwxrwx` lehetnek, de bármelyik helyett állhat `-`. Jelentésük sorra read, write, execute (= olvas, ír, végrehajt), rendre a tulajdonosra, a csoportra, és a többi felhasználóra vonatkoznak. Mindegyik egy bit, ami nulla vagy egy lehet. Most jól jön a nyolcas számrendszer, mivel hármas csoportok vannak. A biteket a

`$ chmod` *fájlok*

paranccsal (change file mode bits) állíthatjuk be. Opciónak egy háromjegyű oktális számot kell megadni: az első bitjei adják meg a tulajdonos jogait, a másodiké a csoport jogait, a harmadiké pedig a „külvilág” jogait. Próbáljuk ki:

```
$ chmod 440 *
```

után az `ls -l` azt mondja, hogy a bitek `r--r-----`, azaz a fájlt a tulajdonos és a csoport olvashatja, más nem, senki nem írhatja, és nem hajthatja végre. Végrehajtási engedélyt csak programoknak értelmes adni. Könyvtáraknak hasonlóan változtathatjuk az engedélyét; itt az `x` bit azt jelenti, hogy a könyvtár listázható. (A `chmod` opciói másként is megadhatók.) Ha most próbáljuk meg a fájlt kitörölni, az `rm` rákérdez, hogy akarjuk-e tényleg törölni az írásvédett fájlt? Válaszoljunk `n`-et. Az `rm` a `-f` (force) opcióval nem kérdez. A fájl tulajdonosát a `chown`, a csoportot a `chgrp` változtatja, míg új csoportot a `newgrp` paranccsal lehet létrehozni, de ezeket nemigen fogjuk használni.

Menjünk a „felső” tesztek könyvtárba és adjunk ki egy

```
$ mv tesztek/rendetlen.txt ./
```

parancsot. Mint látjuk, az írásvédettség nem akadályozza a fájl mozgatását. Hogy miért nem, az később kiderül. Most már kitörölhetjük az „alsó” tesztek könyvtárat.

**1.2.10. Kísérlet.** A rendszerbe a jelszónk segítségével juthatunk be. A rendszergazdától kapott jelszót megváltoztathatjuk, és ajánlatos is megváltoztatni. Ez a

`$ passwd` *loginnév*

paranccsal történhet. A parancs bekéri a régi jelszónkat, majd kétszer az újat. Véletlenszerű jelszót célszerű választani, és olyat, amit minden billentyűzeten be tudunk gépelni (ASCII 33–126 karakterek). Jó, ha legalább 8 karakterből áll, és jobb, ha megjegyezzük, mintha felírjuk. Ha mégis felírjuk, jobb, ha van benne egy 3–4 karakter hosszú rész, amit nem írunk fel. Ez ugyan nem véd meg egy profi ellen, aki látta a felírt részt, de mégse mindenki tud belépni a nevünkben. A felhasználókra vonatkozó információk az `/etc/passwd` fájlban vannak. Ugyanitt, esetleg egy másik fájlban tárolódik a jelszónk is, pontosabban egy lenyomata. Ebből a rendszergazda se tudja visszaállítani a jelszót. Ha elfelejtjük a jelszót, a rendszergazda ki tudja törölni a lenyomatot, és adhat egy új jelszót, amit azonnal megváltoztathatunk.

Ha jobban belegondolunk, elég meglepő, hogy a `passwd` parancs át tudja írni az `/etc/passwd` fájlt. Ha mégnézzük, a fájl tulajdonosa a root. Ha megkeressük a `passwd` parancsot a `/bin` vagy a `/usr/bin` könyvtárban, ahol sok más parancs is van, azt látjuk, hogy a jogokat jelző bitek közül az első `x` helyén `s` áll. Tulajdonképpen van még három bit, amelyek az `x`-ek jelentését módosítják: a legfelső, a setuid az elsőét, a második, a

setgid a másodikét, a harmadik, a sticky bit a harmadikét. Ezek is beállíthatók a `chmod` paranccsal, ha négyjegyű oktális számot adunk meg neki; a legelső jegy jelenti ezeket a biteket. Külön nem íródnak ki, az `x`-ek helyére íródik ki más betű. Az ilyen jogosultságot kapott programok root jogosultságot kapnak, amikor futnak, de mi is használhatjuk őket. Ilyen a `passwd` parancs is. Persze, csak a root tudja állítgatni ezeket a biteket.

Hogy kik vannak még bejelentkezve a gépre, azt a

```
$ who
```

paranccsal nézhetjük meg. Néha szükségünk van a

```
$ whoami
```

parancsra, ha például a `su` (substitute user) paranccsal átjelentkeztünk másnak a nevében, pláne, ha többször is, és már nem tudjuk kinek a nevében dolgozunk. Persze, átjelentkezni csak akkor tudunk, ha ismerjük az illető jelszavát. Még gyakoribb, hogy egy másik gépen vagyunk, és ott más a felhasználói nevünk. Az is előfordulhat, hogy tudjuk a gépünkön a root jelszavát (a root prompt-ja más, `#`). Ekkor sem érdemes azonban root-ként dolgozni: mivel „superman” jogosultságunk van, könnyen elronthatunk bármit, például mindent letörölhetünk. Ha van, inkább használjuk a `sudo` parancsot (substitute user do), amivel csak egy pár parancsot hajthatunk végre root-ként (vagy másként), ha van rá jogosultságunk. Ennek a saját jelszónkat kell megadni.

**1.2.11. Kísérlet.** Hogyan tudjuk távolról használni gépünket, illetve gépünkről egy távoli gépet használni? Erre régebben az `rsh` (remote shell) parancs szolgált, ma a rejtjelezett átvitelt használó `ssh` (secure shell) parancsot használják

```
$ ssh gépnév
```

alakban. A gép hálózati nevét kell megadni, ami általában *gépnév.tartománynév.országnév* alakú, de a tartománynév állhat több ponttal elválasztott részből is. Ha a távoli gépen más a login nevünk, `-l` *loginnév* opcióval azt is meg kell adni. Sok más opció is van, például ablakot is nyithatunk a helyi gépen, de persze ekkor lassabb a munka. Ugyanúgy dolgozhatunk, mintha ott ülnénk a terminál (= végállomás) előtt. A kapcsolatot *Ctrl-d*-vel zárhatjuk le.

Hasonlóan használható az `scp` (secure copy) parancs, mint a `cp`, de a távoli fájl neve elé be kell írni a távoli gép nevét, majd egy kettőspont után az ösvényt. Sokszor egy távoli gépet arra használunk, hogy fontos fájljainkat oda másoljuk megőrzés céljából. Ilyenkor nagyon hasznos az

```
$ rsync könyvtár távoligépnév:távolikönyvtár
```

parancs, rendszerint `-avz` opciókkal. Ez csak annyit másol át, hogy a távoli könyvtár mindent tartalmazzon, ami itt új. A főkönyvtárra is alkalmazható.

**1.2.12. Kísérlet.** Ideje megbeszélni, milyen fájlneveket válasszunk. A UNIX eredetileg 14 betűs fájlneveket engedett meg, ma már hosszabb is lehet a fájlnév. A szokás az, hogy a fájlnév betűvel kezdődik és betűket, számokat és pontot tartalmaz. Betűnek számít az `_` (aláhúzás) is, amelyet a szóköz helyett szokás használni, ezt javasoljuk is. Használhatunk a fájlnévben - jelet is, de az elején nem tanácsos, mivel akkor összekeverhető a fájlnév az opciókkal, amelyek általában akárhol lehetnek. Hagyományosan a

fájl típusát . után 1–3 betű jelzi, bár ez csak szokás. A .txt végződés például szövegfájlt (text) jelent. Mint látjuk, rövidítésekkor a mássalhangzók több információt hordoznak. A . jelentése a fájl elején speciális. A : a fájlnévben gondot okozhat távoli eléréskor. Jól gondoljuk meg, hogy használunk-e ékezetes betűket a fájlnévben: előfordulhat, hogy olyan terminálról kell dolgoznunk, amelyen nincsenek ékezetes betűk. A szóközökkel és ékezetes betűkkel teletűzdelt, kilométer hosszú fájlneveket hagyjuk inkább a titkárnőknek. Persze a speciális karaktereket is jobb elkerülni. Ha elég jól tudunk angolul, legjobb angol neveket használni, ennek az ábécéje a legprimitívebb. A login nevek eredetileg 8 betűsek voltak, ezeket sem érdemes túlcifrázni.

## 1.3. Szövegszerkesztés

## 1.4. További parancsok

**1.4.1. Kísérlet.** tar, gzip

**1.4.2. Kísérlet.** pipeline

# 2. PROGRAMOZÁS

Ennek a résznek a célja ismerkedés a programozással.

# 3. A UNIX SHELL PROGRAMOZÁSA

Ennek a résznek a célja ismerkedés a UNIX shell programozásával.

# 4. A C PROGRAMOZÁSI NYELV ÉS A CWEB

Ennek a résznek a célja ismerkedés a C programozási nyelvvel és a CWEB-bel.

# 5. EGY RISC PROCESSOR TERVEZÉSE

Ennek a résznek a célja ismerkedés egy RISC processzorral és a tervezés teljessé tétele.

# 6. EGY C FORDÍTÓ KÉSZÍTÉSE

Ennek a résznek a célja egy C fordító készítése az előző részben tervezett processzorra.

# 7. EGY MINI UNIX KÉSZÍTÉSE

Ennek a résznek a célja egy mini UNIX készítése az előzőleg tervezett processzorra.

# IRODALOM


[1] Bodi Sándor: *Elektromosság és elektronika. Kézirat.* KLTE, Debrecen, 1987.

[2] Csurgay Árpád – Simonyi Károly: *Elektronfizika.* Tankönyvkiadó, Budapest, 1969.

[3] Feynman, Richard P.: *A felfedezés öröme.* Akkord Kiadó, Budapest, 2002.

[4] LionCommentary on UNIX

[5] Simonyi Károly: *Elektronfizika.* Tankönyvkiadó, Budapest, 1981.

[6] Tietze, U. – Schenk, Ch.: *Analóg és digitális áramkörök.* Műszaki Könyvkiadó, Budapest, 1981.

[7] *Wikipedia.*

[8] Ziel, Aldert van der: *Szilárdtest-elektronika.* Műszaki Könyvkiadó, Budapest, 1982.

# MUTATÓ

A legfontosabb hivatkozás (általában a definíció) oldalszáma *dőlt* betűvel van szedve. A külső, azaz más könyvekre történő hivatkozások előtt szögletes zárójelben szerepel az adott könyv sorszáma az irodalomjegyzékben.