

Fazekas Gábor

Operációs rendszerek

mobiDIÁK könyvtár

Fazekas Gábor

Operációs rendszerek

mobiDIÁK könyvtár

SOROZATSZERKESZTŐ

Fazekas István

Fazekas Gábor

egyetemi docens
Debreceni Egyetem

Operációs rendszerek

Oktatási segédanyag

Első kiadás

mobiDIÁK könyvtár
Debreceni Egyetem

Lektor

Copyright © Fazekas Gábor, 2003

Copyright © elektronikus közlés mobiDIÁK könyvtár, 2003

mobiDIÁK könyvtár

Debreceni Egyetem

Informatikai Intézet

4010 Debrecen, Pf. 12.

Hungary

<http://mobidiak.inf.unideb.hu/>

A mű egyéni tanulmányozás céljára szabadon letölthető. Minden egyéb felhasználás csak a szerző előzetes írásbeli engedélyével történhet. A mű „A mobiDIÁK önszervező mobil portál” (IKTA, OMFB-00373/2003) és a „GNU Iterátor, a legújabb generációs portál szoftver” (ITEM, 50/2003) projektek keretében készült.

Bevezetés

- Mi az operációs rendszer?
- Korai rendszerek.
- A kötegetelt feldolgozás egyszerű rendszerei. (Simple Batch)
- A kötegetelt feldolgozás multiprogramozott rendszerei. (Multiprogramming Batched Systems)
- Időosztásos (time-sharing) rendszerek.
- Személyi számítógépes rendszerek.
- Párhuzamos rendszerek.
- Elosztott rendszerek.
- Valós idejű rendszerek.

- Mi az operációs rendszer?

Operációs rendszer: egy program(rendszer), amely közvetítő szerepet játszik a számítógép felhasználója és a számítógép hardver között.

Operációs rendszer célok:

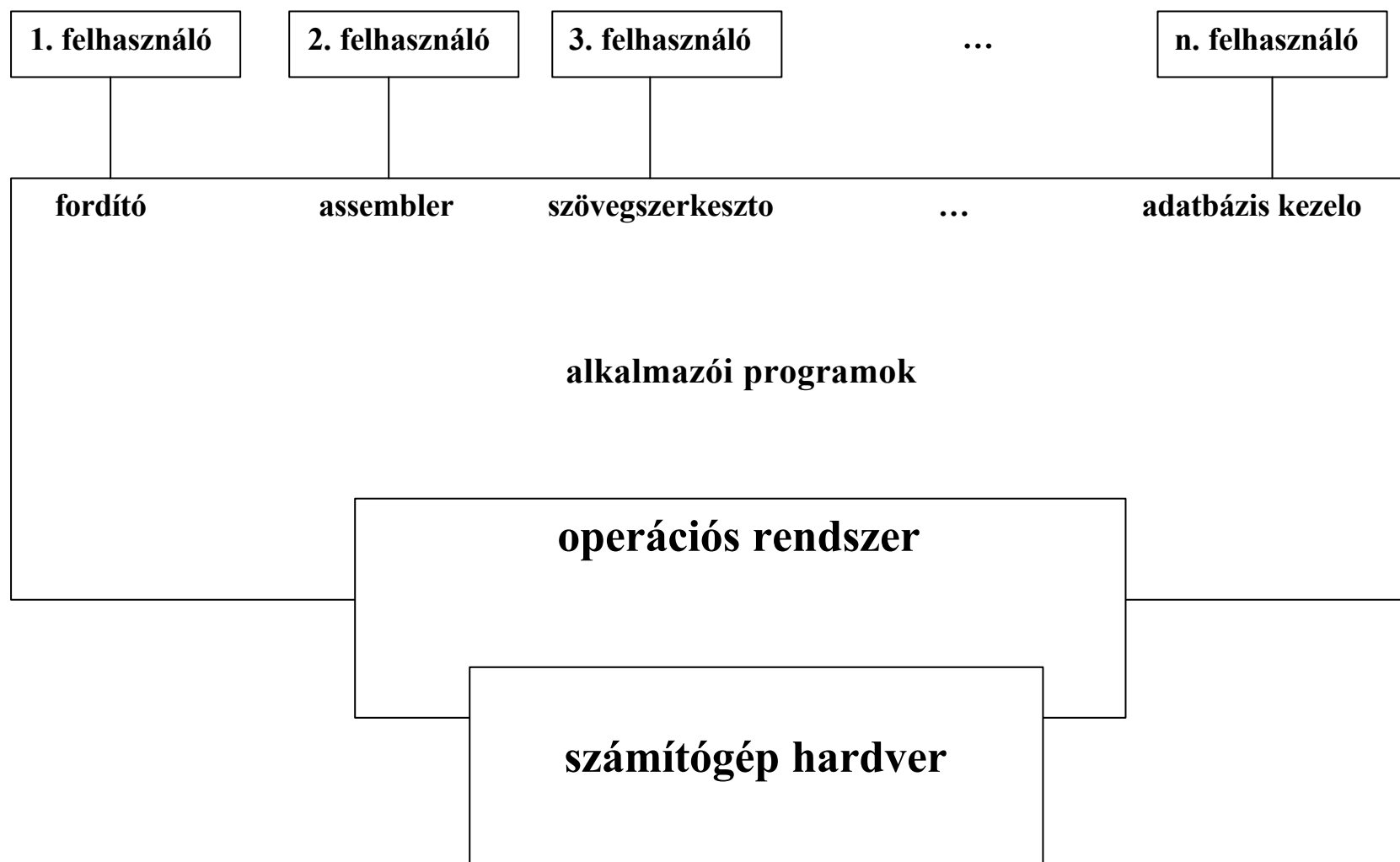
- Felhasználói programok végrehajtása, a felhasználói feladatmegoldás *megkönnyítése*.
- A számítógép rendszer használatának *kényelmesebbé* tétele.
- A számítógép hardver kihasználásának *hatékonyabbá* tétele.

Megjegyzés: az operációs rendszer a felhasználónak *"overhead"*.

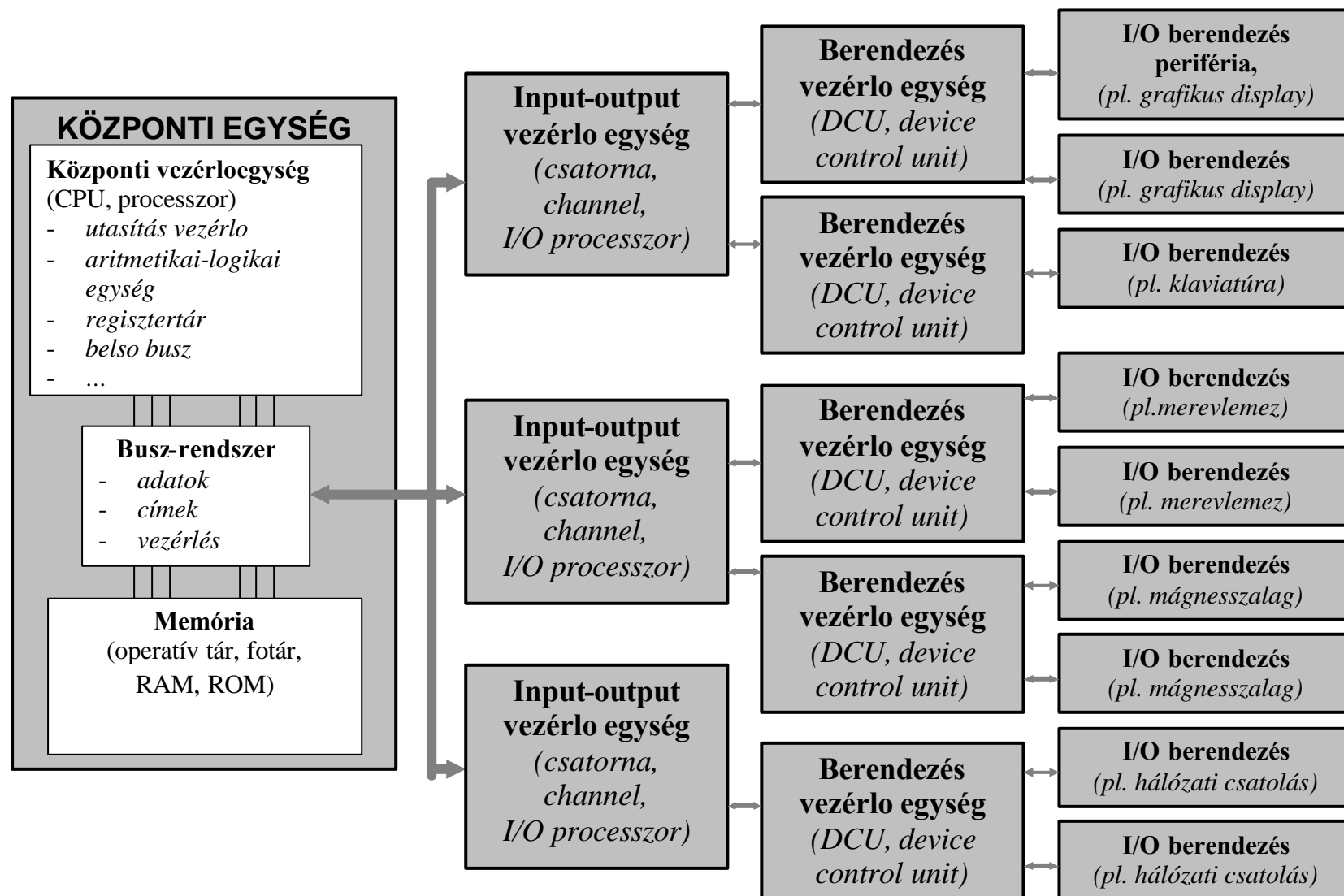
Számítógép rendszerek komponensei (séma)

1. **Hardver** – az alapvető számítási erőforrásokat nyújtja (CPU, operatív memória, I/O berendezések).
2. **Operációs rendszer** – koordinálja és vezérli a hardver erőforrások különböző felhasználók különböző alkalmazói programjai által történő használatát.
3. **Alkalmazói programok** – definiálják azt a módot, ahogyan az egyes rendszer-erőforrásokat a felhasználók számítási problémáinak megoldásához föl kell használni (fordítók, adatbázis kezelők, videó játékok, ügyviteli programok).
4. **Felhasználók** (emberek, gépek, más számítógépek).

Számítógép rendszerek komponensei (séma)



A számítógép funkcionális felépítése



Operációs rendszer definíciók

... nézetfüggő :

- ***Erőforrás allokáló***/kiosztó – menedzseli és kiosztja a hardver erőforrásokat
- ***Felügyelő program*** – felügyeli a felhasználói programok végrehajtását, az I/O berendezések működését.
- ***Kernel (mag)*** – az egyetlen program, amelyik "állandóan fut" (minden más program alkalmazói program).

- Korai rendszerek – "pucér" gép (1950-es évek eleje)
 - Szerkezeti jellemzők
 - a nagyméretű gépet a *konzolról* irányítják,
 - egy felhasználós rendszer,
 - a programozó egyben operátor is,
 - lyukszalagos és/vagy lyukkártyás adatbevitel és kivitel.
 - Korai szoftver
 - assemblerek,
 - betöltők (loaderok),
 - kapcsolat szerkesztők (linkage editor),
 - közös szubrutin-könyvtárak,
 - fordítók (compiler-ek),
 - I/O berendezés kezelő rutinok (device driver-ek).
 - Biztonság
 - Drága erőforrások rossz hatékonyságú kihasználása
 - alacsony CPU kihasználtság,
 - jelentős mennyiségű "beállítási idő" (setup time).

- A kötegelte feldolgozás rendszerei (Simple Batch) I.
 - Vegyünk fel egy (professzionális) operátort.
 - Felhasználó ? operátor.
 - Adjunk a rendszerhez egy kártyaolvasót.
 - Redukáljuk a *beállítási időt* a (használt) munkák (job) kötegelésével.
 - Automatikus soros munka végrehajtás (job sequencing): a vezérlés egyik jobról (a job vége után) automatikusan kerül át a következőre. (Az első elemi operációs rendszer megjelenése).
 - Rezidens monitor (felügyelőprogram) működési elve:
 - kezdetben a vezérlés a monitornál van,
 - a vezérlés átadódik a job-nak,
 - ha a job befejeződött a job vissza kerül a monitorhoz.

• A kötegetelt feldolgozás rendszerei (Simple Batch) II.

Egy tipikus job szerkezete és a számítógépes problémamegoldás folyamata, job lépés (jobstep).

Problémák:

1. Hogyan szerezhethet a monitor tudomást az adott job természetéről (pl. FORTRAN vagy ASSEMBLY), vagy melyik programot kell végrehajtani?
2. Hogyan tudja a monitor megkülönböztetni
 - egyik job-ot a másiktól?
 - az adatot a programtól?

Megoldás: Vezérlő kártyák, pozicionálás

- Speciális kártyák, amelyek megmondják a monitornak, mely programot kell futtatni (\$JOB, \$FTN, \$RUN, \$DATA, \$END)
- Speciális karakterek különböztetik meg az adat és program kártyákat. (//, \$, 7-2 lyukasztás)

• A kötegetelt feldolgozás rendszerei (Simple Batch) III.

- A rezidens monitor funkcionális részei
 - Vezérlő kártya interpreter – felelős a vezérlőkártyák beolvasásáért és értelmezéséért.
 - Betöltő (loader) – háttértárból betölti az egyes rendszer és felhasználói programokat az operatív memóriába.
 - Készülék meghajtó programok (device drivers) – ismerik a rendszer az egyes I/O berendezéseinek tulajdonságait és működtetésük logikáját.
- Előny: csökken a beállítási idő (setup time)!

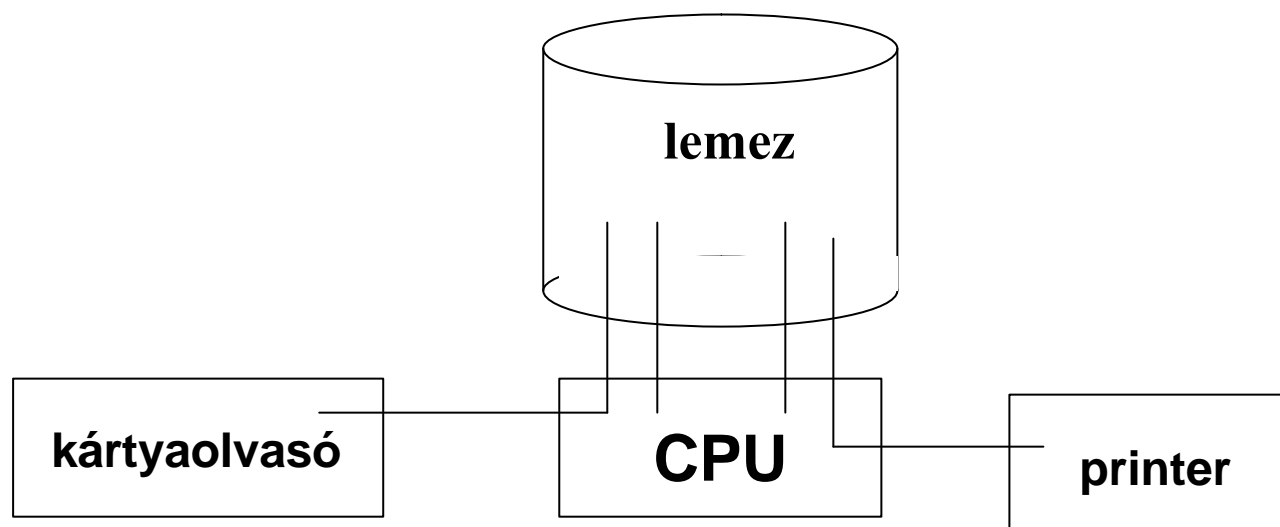
Probléma: Alacsony teljesítmény – mivel az I/O és a CPU műveletek nem fedhetik át egymást (párhuzamosság!) és a kártyaolvasó nagyon lassú!

Megoldás: Off-line elő- és utófeldolgozás – a jobokat egy másik gép segítségével szalagra másoljuk, ill. az eredményeket szalagra írjuk, majd egy másik gép nyomtatja ki. (Absztrakt periféria fogalom igénye megfogalmazódik!)

Simultaneous Peripheral Operation On-Line (SPOOL): IBM704, 1960...

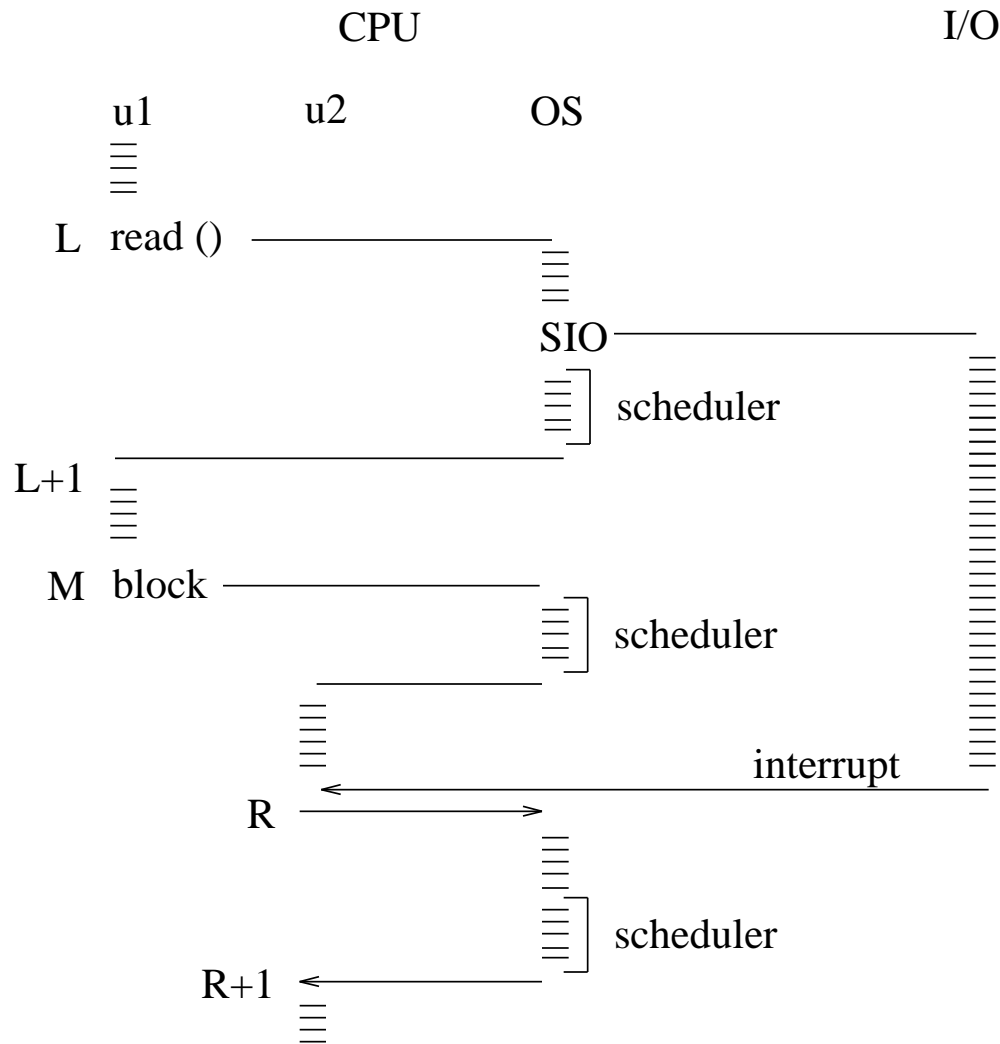
A kötegelt feldolgozás rendszerei (Simple Batch) IV.

Még jobb megoldás: **Spooling!**



- Mialatt egy job végrehajtódik, az operációs rendszer:
 - beolvassa a következő jobot a kártyaolvasóról a lemezre (job queue)
 - egy előző job által nyomtatni szánt adatokat lemezről printerre továbbítja
- Job pool – olyan adatszerkezet, amelynek segítségével az operációs rendszer kiválaszthatja a következő job-ot, CPU kihasználtság növelése.

- A kötegetelt feldolgozás multiprogramozott rendszerei.
(Multiprogramming Batched Systems)
 - Alapelv: néhány job(step) futtatható kódja állandóan az operatív memóriában helyezkedik el és készen áll arra, hogy “utasításokkal lássa el” a CPU-t.
 - Vigyázat! Nem “párhuzamosan futó” programokról van szó!
 - Az operációs rendszer valamilyen *stratégia* szerint adja oda a CPU-t a futásra kész programoknak.



- A multiprogramozás által az operációs rendszerekkel szemben támasztott követelmények
- Az I/O-nak az operációs rendszer részéről történő teljes körű felügyelete. (adatvédelem!)
 - Az I/O-t az operációs rendszer nem egyszerűen támogatja, hanem végrehajtásához elkerülhetetlen.
 - Hardver feltételek! (kernel/supervisor mode, privileged operations)
- Memória gazdálkodás – a rendszernek fel kell osztania a memóriát a futó jobok között.
 - Hardver feltételek! (kernel/supervisor mode, privileged operations, segmentation)
- CPU ütemezés – a rendszernek választani kell tudni a futásra kész jobok között.
- Készülékhozzárendelés
 - Nem “jut” minden jobnak, printer, lemez, stb.
- Időosztásos (time-sharing) rendszerek – interaktivitás

- A kötegelt rendszerek hátránya: nincs interaktivitás!
- TS esetén a CPU váltakozva áll olyan joboknak a rendelkezésére, amelyek a memóriában, vagy lemezen találhatók. (Természetesen a CPU-t csak olyan job kaphatja meg, amely éppen a memóriában van.)
- Egy job a lemezről a memóriába, ill. a memóriából a lemezre betölthető/kimenthető az ütemezési stratégiának (időosztás!) megfelelően. Új fogalom: folyam (process)!
- A rendszer és a felhasználó között on-line kommunikációt tételezünk fel; ha az operációs rendszer befejezi egy parancs végrehajtását, a következő “vezérlő utasítás”-t nem a kártyaolvasóról, hanem a felhasználó klaviatúrájáról várja.
- Egy – adatokat és utasításkódokat tároló – on-line fájl-rendszer kell, hogy a felhasználók rendelkezésére álljon.

- Személyi számítógépes rendszerek.
- Személyi számítógépek – a teljes számítógép rendszer egy egyszerű felhasználónak kizárólagos rendelkezésére áll.
- Tipikus konfigurációjú I/O berendezések – klaviatúra, egér, képernyő kijelző, kis teljesítményű nyomtató.
- Előtérben a felhasználó (személy) kényelme és felelőssége.
- Sokszor adaptál – eredetileg nagygépes operációs rendszerekre kidolgozott információ technológiai megoldásokat. (migráció!)
 - példa: MULTICS (MIT, 1965-70) \Rightarrow UNIX (Bell Labs, 1970)
- A felhasználó személy sokszor a számítógép kizárólagos tulajdonosa, felhasználója, és így nincs szüksége fejlett CPU kiszolgáló és adatvédő szolgáltatásokra.

- Párhuzamos rendszerek – multiprocesszoros rendszerek több mint egy – szoros kommunikációs kapcsolatban levô – CPU-val
- Szorosan kapcsolt/csatolt rendszerek – a processzorok közösen használják a memóriát és a rendszer órát. A kommunikáció a közös memória segítségével történik.
- Párhuzamos rendszerek előnyei:
 - Megnövelt átbocsátó képesség,
 - Gazdaságosság,
 - Növekvô megbízhatóság,
 - Redundancia,
 - Graceful degradation,
 - Fail-soft rendszerek.

• Párhuzamos rendszerek

• Szimmetrikus multiprocessing

- Minden egyes processzor az operációs rendszer azonos változatát (másolatát) futtatja. Ezek egymással szükség szerint kommunikálhatnak.
- Sok processzus futhat egyszerre teljesítménycsökkenés nélkül.
 - I/O problémák, ütemezés

• Aszimmetrikus multiprocessing (master-slave modell)

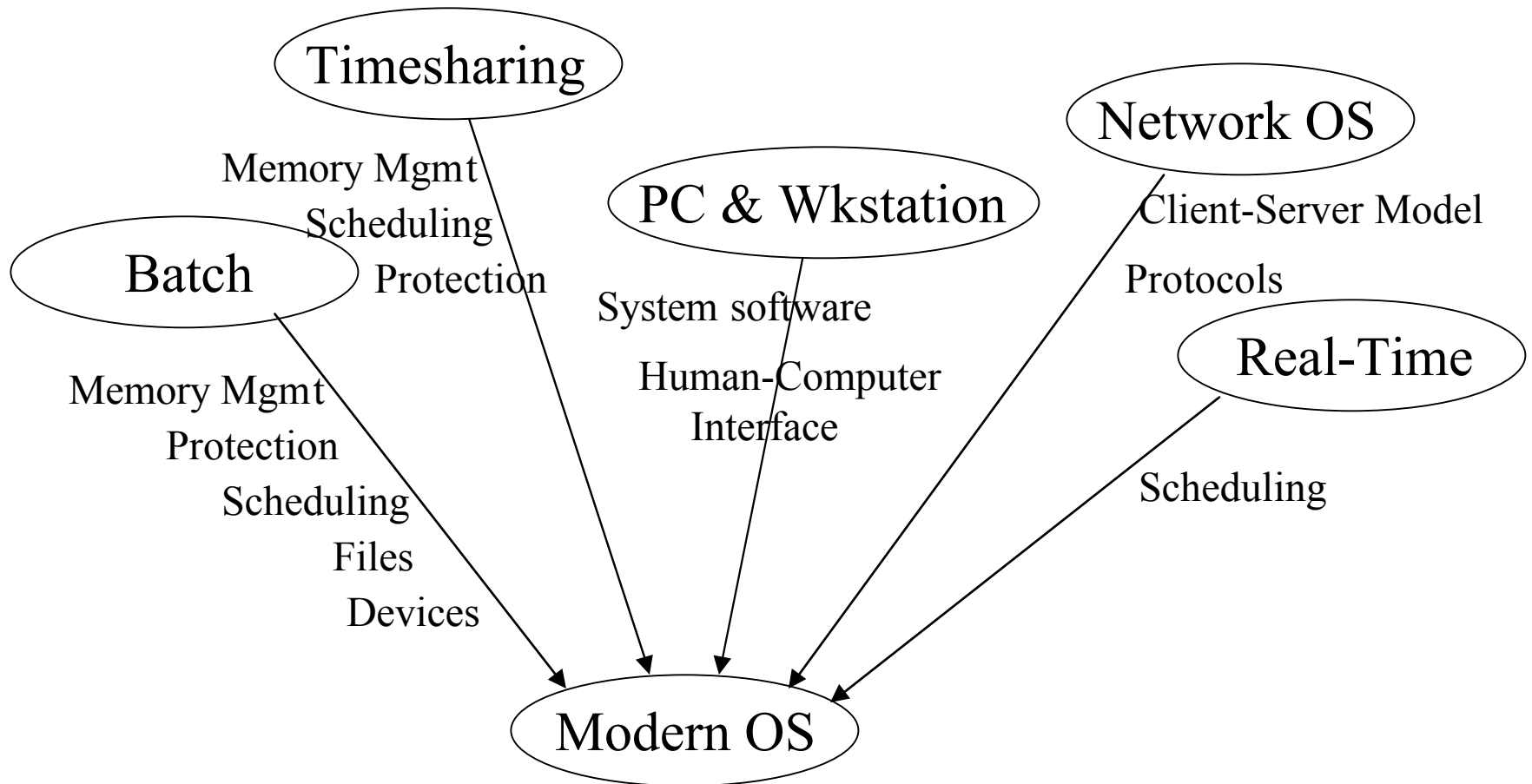
- Minden egyes processzor a hozzárendelt specifikus feladatot (task) oldja meg. A feladatot a mester határozza meg! Ezek a taskok egymással szükség szerint kommunikálhatnak.
- Nagyon nagy rendszerekben elterjedtebb megoldás.
 - RJE (remote job entry), front-end processzorok.

- Elosztott rendszerek – a számításokat több processzor között osztják meg.
- *Lazán kapcsolt/csatolt rendszerek* – a processzorok saját lokális memóriát és rendszer órát használnak. A kommunikáció nagy kapacitású adatvonalak, vagy telefon vonalak segítségével történik.
 - (Pl. speciális LAN; új fogalmak: site, node)
- Elosztott rendszerek előnyei:
 - Erőforrás megosztás (printer, stb.),
 - Számítási teljesítmény növelés,
 - túlterhelés védelem (load sharing),
 - Növekvő megbízhatóság,
 - Kommunikáció (e-Mail, stb).

• Valós idejű (real-time) rendszerek

- Gyakran úgy jelenik meg, mint valamilyen dedikált alkalmazás (pl. tudományos kísérlet támogatása, orvosi képfeldolgozás, ipari kontroll, kijelző rendszerek) irányító-felügyelő rendszere.
- A “kiszolgálás” azonnal megkezdődik! Jól definiált, rögzített idejű korlátozások.
- “Hard” (“merev” valós idejű) rendszerek.
 - A másodlagos tár korlátozott, vagy teljesen hiányzik; az adatokat az operatív memóriában (RAM), vagy akár ROM-ban tárolják.
 - Konfliktus az időosztásos rendszerekkel.
- “Szoft” (“puha” valós idejű) rendszerek.
 - Korlátozott szolgáltató programok az ipari kontroll, a robotika területén.
 - A fejlett operációs rendszer szolgáltatásokat igénylő alkalmazásoknál (Multimédia, VR) igen hasznosak.

Evolution of Modern OS



2. A számítógép rendszer strukturális jellemzői

- A számítógépes rendszer működése
- I/O struktúra
- Tár struktúra és hierarchia
- Hardver védelem
- Általános rendszer architektúra

- A számítógépes rendszer működése
 - Az I/O berendezések és a CPU szimultán képes működni.
 - Minden berendezés vezérlő egység egy meghatározott berendezés típus működtetéséért felelős,
 - Minden berendezés vezérlő egységnek saját, lokális puffere van.
 - A CPU oldaláról az I/O a főtár és e lokális pufferek közötti adatmozgatást jelenti.
 - A berendezés vezérlő egységek, I/O processzorok egy-egy megszakítás generálásával jelezhetik a CPU-nak az I/O művelet befejezését. (DMA)

• A megszakításkezelés fogalmai

- A *megszakítás (interrupt)* átadja a vezérlést a *megszakítás feldolgozó rutinnak*. Ez általában a *megszakítási vektor* segítségével történik, amelynek megfelelő elemei tartalmazzák a *megszakítási osztályokhoz* tartozó feldolgozó rutin első végrehajtandó utasításának címét.
- A megszakítási rendszernek tárolnia kell a megszakított utasítás címét.
- Egy megszakítás feldolgozása idején jelentkező további megszakítások letilthatók (mask out, disable), hogy el ne "vesszenek" (lost interrupt).
- A megszakítási jel forrását tekintve egy megszakítás lehet külső (pl. I/O, Timer, Hardver, ...), vagy belső (szoftver megszakítás, angolul: *trap*).
- Az operációs rendszer (megszakítás feldolgozó rutin) közvetlen feladatai:
 - a CPU állapotának megőrzése (a hardver ehhez minimális kezdeti támogatást nyújt);
 - a megszakítás okának, körülményeinek részletesebb elemzése;
 - "maszkolás";
 - a megszakított programhoz történő "visszatérés" megszervezése.

• I/O struktúra

- Az I/O folyamat elindult, a felhasználói program a vezérlést csak az I/O befejezése után kapja vissza.
 - "wait" utasítások a következő megszakításig;
 - "wait loop" az adat megérkezéséig (hand-shaking, ready bit!);
 - csak egyetlen egy I/O lehet egyszerre folyamatban.
- Az I/O folyamat elindult, a (felhasználói) program a vezérlést azonnal vissza-kapja tekintet nélkül az I/O befejeződésére.
 - rendszer hívás (system call); az I/O-t a felügyelő program indítja;
 - készülék - státusz táblázat (device state);
 - a felügyelő program kezeli - gazdálkodás, ütemezés.
- DMA (Direct Memory Access)
 - egy megszakítás blokkonként (nem bájtonként!)

• Tár struktúra és hierarchia

- Fôtár, operatív tár – az egyetlen, amit a CPU közvetlenül képes elérni.
- Másodlagos tár – a fôtár *kiterjesztése*, nagy, nem törlődô tárolás lehetősége.
 - Swap, paging.
- Mágnes lemezek
 - sáv, szektor, cylinder fogalma
- A tár rendszerek hierarchikus struktúrába rendezhetők
 - sebesség,
 - ár (költség),
 - a tárolt adatok tartósságaalapján.
- Caching, cache szervezés:
 - belső/külső cache, virtuális lemezek.

• Hardver védelem

• Duál - módú működés

- felhasználó mód – rendszer (supervisor, monitor) mód,
- módus bit,
- privilegizált műveletek.

• I/O védelem

- minden I/O utasítás privilegizált
- meg kell(ett) oldani, hogy felhasználói programok ne kerülhessenek monitor módba

• Memória védelem

- szegmensek, tárvédelmi kulcs
- monitor módban korlátlan a tárhozzáférés

• CPU védelem

- Timer megszakítások szerepe

- **Általános rendszer architektúra**
- Az I/O utasítások privilegizáltak. Hogyan hajthat végre egy felhasználói program I/O műveletet?
- Rendszer hívás - speciális megszakítás az operációs rendszer szolgáltatásainak igénybe vételére.
 - INT gépi utasítás,
 - paraméter átadás, funkció kódok,
 - paraméter ellenőrzés ,
 - végrehajtás,
 - vezérlés visszaadása.

3. Operációs rendszer struktúrák

- Az operációs rendszer komponensei
- Operációs rendszer szolgáltatások
- Rendszer-hívások
- Rendszerprogramok
- Rendszer-szerkezet
- Virtuális gépek
- Rendszer tervezés és implementáció
- Rendszer-generálás

- Az operációs rendszer komponensei (idealizált)
 - Folyamat kezelés (Process management)
 - Memória kezelés (gazdálkodás)
 - Másodlagos tár kezelés
 - I/O rendszer kezelés
 - Fájl kezelés
 - Védelmi rendszer
 - Hálózat-elérés támogatása
 - Parancs interpreter rendszer

- Folyamat kezelés (Process management)
 - Folyamat (process - processzus) – egy végrehajtás alatt levő program. A folyamatnak bizonyos erőforrásokra (így pl. CPU idő, memória, állományok, I/O berendezések) van szüksége, hogy a feladatát megoldhassa.
 - Az operációs rendszer az alábbi tevékenységekért felel a folyamatok felügyeletével kapcsolatban:
 - Folyamat létrehozása és törlése.
 - Folyamat felfüggesztése és újraindítása.
 - Eszközök biztosítása a
 - folyamatok szinkronizációjához,
 - a folyamatok kommunikációjához.

- Memória (fôtár) kezelés (gazdálkodás)
 - Az operatív memóriát bájtokból (szavakból) álló tömbnek fogjuk tekinteni, amelyet a CPU és az I/O vezérlô megosztva (közösén) használ.
 - Tatalma törlôdik a rendszer kikapcsolásakor és rendszerhibáknál.
 - Az operációs rendszer a következô dolgokért felelôs a memória kezelést illetôen:
 - Nyilvántartani, hogy az operatív memória melyik részét ki (mi) használja.
 - Eldönteni, melyik folyamatot kell betölteni, ha memória felszabadul.
 - Szükség szerint allokálni és felszabadítani memória területeket a szükségleteknek megfelelôen

- Másodlagos tár kezelés

- Mivel az operatív tár (elsődleges tár) törlődik (és egyébként sem alkalmas arra, hogy minden programot tároljon), a másodlagos tárra szükség van.
- Merev lemezes tár, a másodlagos tár legelterjedtebb megjelenése.
- Az operációs rendszer a következő dolgokért felelős a másodlagos tár kezelést illetően:
 - Szabad-hely kezelés
 - Tár-hozzárendelés
 - Lemez elosztás (scheduling)

- I/O rendszer kezelés

- Az I/O rendszer az alábbi részekből áll:
 - Puffer (Buffer/Cache) rendszer
 - Általános készülék-meghajtó (device driver) interface
 - Speciális készülékek meghajtó programjai

• Fájl (állomány) kezelés

- Egy fájl kapcsolódó információ (adatok) együttese, amelyet a létrehozója definiál. Általában program- (különböző formák), vagy adatfájlokkal dolgozunk.
- Az operációs rendszer a következő dolgokért felelős a fájl kezelést illetően:
 - Fájl létrehozása és törlése
 - Könyvtár létrehozása és törlése
 - Fájlokkal és könyvtárakkal történő alap-manipulációhoz nyújtott támogatás.
 - Fájlok “leképezése” a másodlagos tárba.
 - Fájlok mentése valamilyen nemtörlődő, stabil adathordozóra.

- Védelmi rendszer
- Védelem általában valamilyen mechanizmusra utal, amelynek révén mind a rendszer-, mind a felhasználói erőforrásoknak a programok, folyamatok, vagy felhasználók által történő elérése felügyelhető, irányítható.
- A védelmi mechanizmusnak tudnia kell:
 - különbséget tennie autorizált (jogos) és jogtalan használat között,
 - specifikálni az alkalmazandó kontrolt,
 - szolgáltatni a korlátozó eszközöket.

- Hálózat-elérés támogatása (elosztott rendszerek)
 - Egy elosztott rendszer processzorok adat és vezérlő vonallal összekapcsolt együttese, ahol a processzorokhoz nincs közös memóriájuk és órájuk. (lokális memória, óra).
 - Az adat- és vezérlő vonalak egy kommunikációs hálózat részei.
 - Az elosztott rendszer a felhasználóknak különböző osztott erőforrások elérését teszi lehetővé.
 - Az erőforrások osztott elérése lehetővé teszi:
 - a számítások felgyorsítását,
 - a jobb adatelérhetőséget,
 - a nagyobb megbízhatóságot.

• Parancs interpreter (al)rendszer

- Az operációs rendszernek sok parancsot ún. vezérlő utasítás formájában lehet megadni. Ezek a vezérlő utasítások az alábbi területekhez tartozhatnak:
 - folyamat létrehozás és kezelés
 - I/O kezelés
 - másodlagos tár kezelés
 - operatív tár kezelés
 - fájl rendszer elérés
 - védelem
 - hálózat kezelés
 - ...
- Az operációs rendszernek azt a programját, amelyik a vezérlő utasításokat (be)olvassa és interpretálja a rendszertől függően más és más módon nevezhetik:
 - vezérlő kártya interpreter (control-card Job Control, JC)
 - parancs-sor interpreter (command-line)
 - héj (burok, shell) (UNIX)

• Operációs rendszer szolgáltatások

- Program végrehajtás (program betöltés és futtatás)
- I/O műveletek (fizikai szint: blokkolás, pufferezés)
- Fájl-rendszer manipuláció (r, w, c, d)
- Kommunikáció – a folyamatok közötti információ csere (ugyanazon, vagy különböző gépeken) Shared memory – Message passing.
- Hiba detektálás (CPU, memória, I/O készülékek, felhasználói programok, ...)
- Nem közvetlenül a felhasználó támogatását, hanem a hatékonyabb rendszerműködést segítik:
 - Erőforrás kiosztás – multiprogramozás, többfelhasználós működés
 - Accounting – rendszer és felhasználói statisztikák.
 - Védelem – minden erőforrás csak az operációs rendszer felügyelete mellett érhető el.

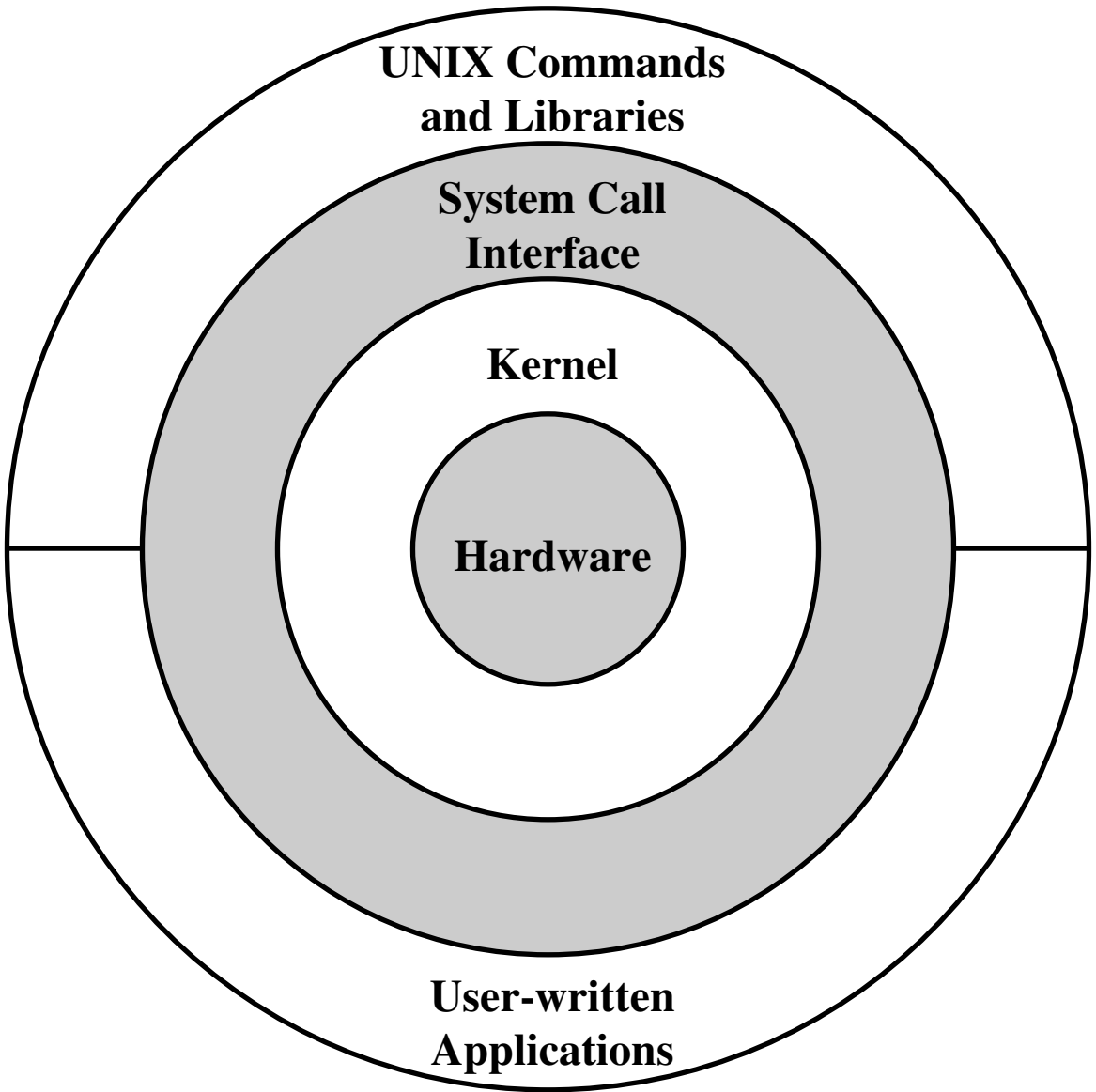
• Rendszer-hívások

- Ha a futó (felhasználói) program valamilyen rendszerszolgáltatást igényel, ezt rendszerhívás formájában teheti meg.
 - Általában assembly szintû utasításként jelen van az architektúrában. (INT, Tcc, SC, stb.)
 - Magas szintû, rendszerprogramok írására szánt programozási nyelvekbe is beépítették. (C)
- Paraméterátadás módjai a rendszernek:
 - regiszterben,
 - paraméter-táblázatban,
 - veremben (push- felh. program; pop- op. rendszer) ,
 - a módszerek kombinálása, statusword-ok

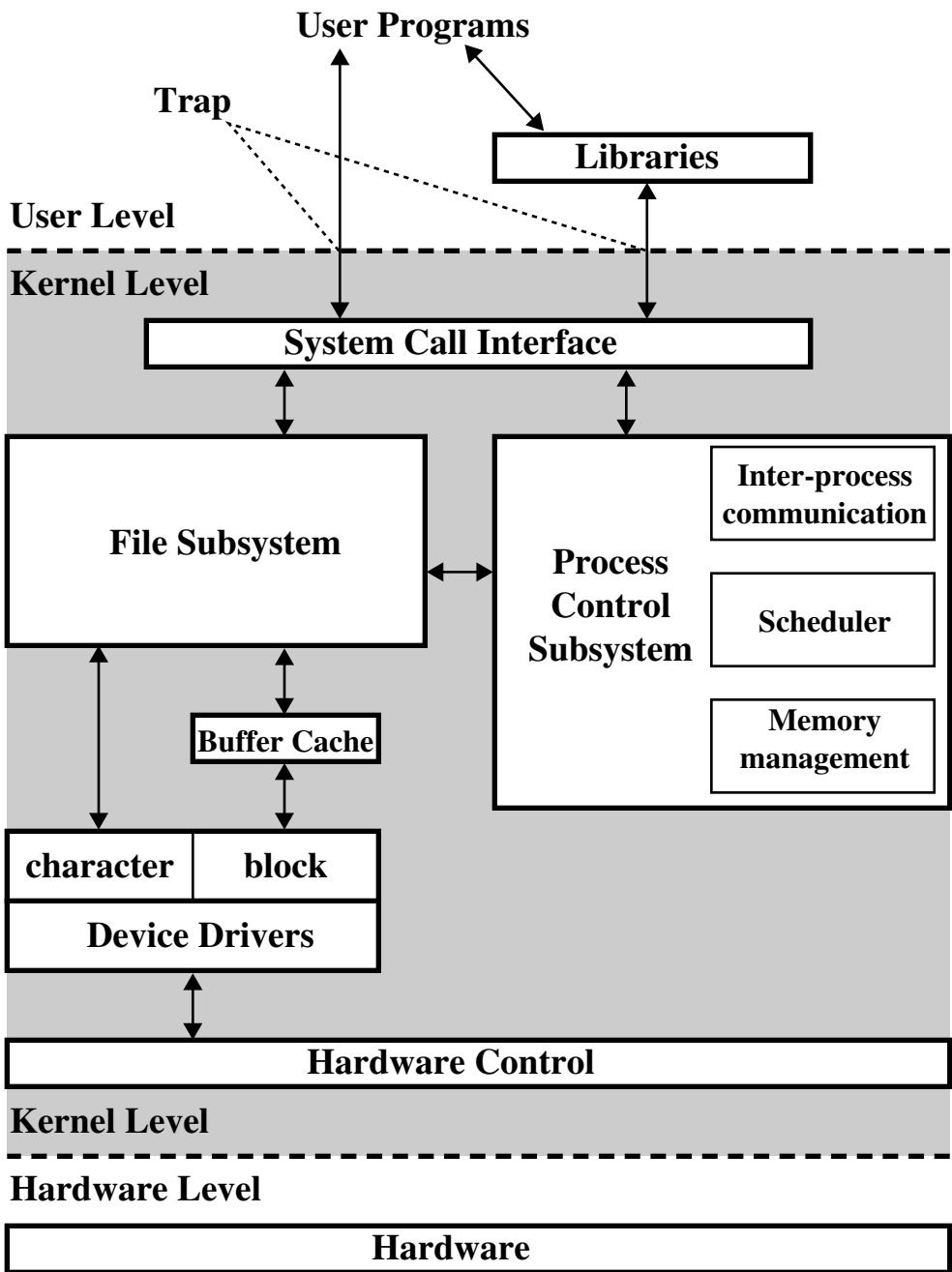
- **Rendszerprogramok**
- A rendszerprogramok kényelmes környezetet teremtenek a program-fejlesztéshez és program végrehajtáshoz. Egy lehetséges osztályozásuk:
 - Fájl manipuláció
 - Státusz információ
 - Fájl módosítás
 - Programozási nyelv támogatás
 - Program betöltés és végrehajtás
 - Kommunikáció
 - Alkalmazói programok ...
- Felhasználói szempontból nézve (egyes nézetek szerint) az operációs rendszer sokszor a rendszerprogramok együttesével azonosítható.

• Rendszer-szerkezet

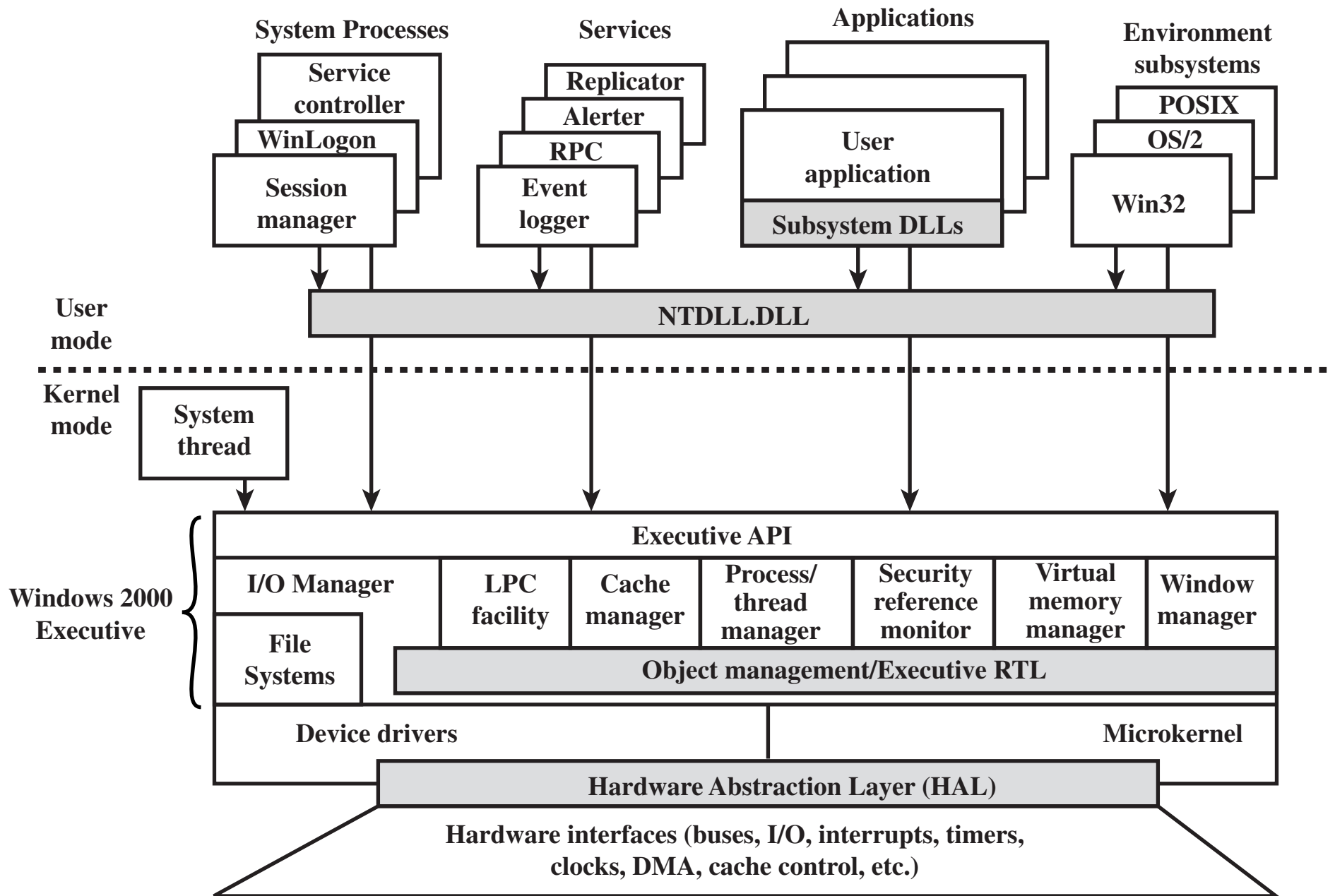
- **MS-DOS** – *a lehető legtöbb funkcionalitást igyekszik besűríteni a lehető legkisebb tárba:* (ezért) nincs modularitás. Az MS-DOS -nak van bizonyos szerkezete, de kapcsolatai és funkcionális szintjei nem különböztethetők meg élesen. (\Rightarrow OS/2)
- **UNIX** – a hardver funkcionalitása a korlát; az eredeti UNIX operációs rendszer korlátozott struktúrával rendelkezett:
 - rendszerprogramok,
 - a kernel (mag), amely mindent magában foglal a rendszerhívás interfész alatt és a fizikai hardver fölött: (pl.) fájl rendszer, CPU ütemezés, memória gazdálkodás, más operációs rendszer függvények, azaz számos függvény egy szinten.



General UNIX Architecture



Traditional UNIX Kernel]



Windows 2000 Architecture

- Rétegelt (layered) megközelítés

- Az operációs rendszer egymásra épülő szintekre bomlik. Legalsó szinten (layer 0) van a hardver, a legfelsőn (layer N) a felhasználói interfész.
- Megfelelő modularitással minden szint (layer) kizárólag a nála alacsonyabb szint függvényeit és szolgáltatásait használja.
- Ilyen absztrakt rétegelt megközelítést először *Dijkstra* javasolt a *T.H.E* operációs rendszerben:
 - 5. Szint: felhasználói program
 - 4. Szint: input/output pufferezés
 - 3. Szint: operátor konzol készülék meghajtó
 - 2. Szint: memória menedzsment
 - 1. Szint: CPU ütemezés
 - 0. Szint: hardver

• Virtuális gépek

- A virtuális gép a rétegelt modell általánosítása: nemcsak a fizikai hardvert, hanem az operációs rendszer magját is hardvernek tekinti.
- A virtuális gép a rendelkezésre álló hardverrel azonos interfészt nyújt.
- Az operációs rendszer azt az illúziót kelti, mintha minden processzus saját processzort és saját (virtuális) operatív memóriát használna.
- A fizikai gép erőforrásainak megosztásával (többszörös felhasználásával) implementálhatók a virtuális gépek:
 - CPU ütemezés \Rightarrow saját processzorok illúziója
 - SPOOLING és fájl rendszer \Rightarrow saját perifériák illúziója
 - Valódi időosztásos terminál \Rightarrow a virtuális gép operátori konzolja
- A virtuális gépek előnyei és hátrányai
 - Izoláció \Rightarrow teljes védelmet nyújt, de kizárja az erőforrások célszerű közös használatát.
 - Jó közeg a rendszerprogramozónak: teljes szolgáltatás a környezet zavarása nélkül.
 - Bonyolult feladat az implementáció (a valódi gép pontos multiplikálása).

• Rendszer tervezés és implementáció

• Rendszer tervezési célok

- **Felhasználói célok:** az operációs rendszer legyen kényelmesen használható, könnyen megtanulható, megbízható, biztonságos, gyors.
- **Rendszer célok:** az operációs rendszer legyen könnyen tervezhető, implementálható, gondozható; továbbá rugalmas, megbízható, hiba-mentes, hatékony.

• Mechanizmusok és politikák

- A mechanizmusok meghatározzák, hogy **hogyan** kell valamit csinálni, a politikák pedig, hogy **mit** kell csinálni. (példa: timer megszakítás.)
- A mechanizmus és a politika különválasztása nagyon fontos; maximális rugalmasságot teremtet, ha a politika később megváltozik.

• Rendszer implementálás

- Régen assembly nyelven írták az operációs rendszereket, ma már ez nem igaz; magasszintű nyelven is megírhatók. (pl.: MCP/Borroughs \Rightarrow Algol; MULTICS \Rightarrow PL/1; UNIX, OS/2, WinNT \Rightarrow C)
- A magasszintű nyelven írt kód gyorsabban elkészíthető, kompaktabb, könnyebben áttekinthető és nyomkövethető (debug).
- A magasszintű nyelven írt rendszer portális, más hardverre könnyen átvihető. (pl.: Unix, Linux) A szűk keresztmetszeteket meghatározva a gépi kód korrigálható (patch, service pack).

Rendszer-generálás

- Az operációs rendszereket úgy tervezik, hogy azok működőképesek legyenek egy géposztály minden gépén. Ehhez azonban minden egyes számítógép-környezetre az operációs rendszert *konfigurálni* kell. Ezt a műveletet *rendszer-generálásnak* (\Leftrightarrow installálás?) nevezzük. Program segíti (SYSGEN, setup, install).
- Egy SYSGEN program informálódik a hardver rendszer specifikus konfigurációjáról. (pl. Milyen processzor, aritmetika vehető alapul; processzorok száma, típusa, stb. ; rendelkezésre álló memória mérete; perifériális berendezések típusai; megszakítási rendszer tulajdonságai).
- Tisztázni kell, hogy a rendszer mely szolgáltatásai tényleg szükségesek: multiprogramozási stratégia, hálózat elérés, stb.
- Booting, bootstrap loader – rendszer betöltés.

•Rendszer-generálás

A generált rendszerről szerzett adatok (UNIX példa)

G E N E R A L I N F O R M A T I O N

```
Host Name is                euklid
Host Address(es) is         131.234.xxx.xxx
Host ID is                   80782854
Serial Number is             2155358292
Manufacturer is              Sun (Sun Microsystems)
System Model is              Ultra 1 Model 140
Main Memory is               128 MB
Virtual Memory is            210 MB
ROM Version is               OBP 3.0.4 1995/11/26 17:47
Number of CPUs is            1
CPU Type is                  sparcv8plus+vis
App Architecture is          sparc
Kernel Architecture is       sun4u
OS Name is                   SunOS
OS Version is                 5.6
OS Distribution is           Solaris 2.6 5/98 s297s_hw3smccServer_09 SPARC
Kernel Version is            SunOS Release 5.6 Version Generic_105181-17
                               [UNIX(R) System V Release 4.0]
Boot Time is                 Sat Feb 26 18:39:29 2000
```

K E R N E L I N F O R M A T I O N

Maximum number of processes for system is	1034
Maximum number of processes per user is	1029
Maximum number of users (for system tables) is	64
Maximum number of BSD (/dev/ptyXX) pty's is	48
Maximum number of System V (/dev/pts/*) pty's is	48
Size of the virtual address cache is	16384
Size of the callout table is	112
Size of the inode table is	4712
Size of the directory name lookup cache is	4712
Size of the quotas table is	1674
STREAMS: Maximum number of pushes allowed is	9
STREAMS: Maximum message size is	65536
STREAMS: Maximum size of ctl part of message is	1024
Maximum global priority in sys class is	6488124
Has UFS driver is	TRUE
Has NFS driver is	TRUE
Has SD driver is	TRUE
Has FD driver is	TRUE
Has IPCSHMEM is	TRUE

S Y S C O N F I N F O R M A T I O N

Max combined size of argv[] and envp[] is	1048320
Max processes allowed to any UID is	1029
Clock ticks per second is	100
Max simultaneous groups per user is	16
Max open files per process is	64
System memory page size is	8192
Job control supported is	TRUE
Savid ids (seteuid()) supported is	TRUE
Version of POSIX.1 standard supported is	199506
Version of the X/Open standard supported is	3
Max log name is	8
Max password length is	8
Number of processors (CPUs) configured is	1
Number of processors (CPUs) online is	1
Total number of pages of physical memory is	16384
Number of pages of physical memory not currently in use is	7823
Max number of I/O operations in single list I/O call is	256
Max number of timer expiration overruns is	2147483647
Max number of open message queue descriptors per process is	32
Max number of message priorities supported is	32
Max number of realtime signals is	8
Max number of semaphores per process is	2147483647
Max value a semaphore may have is	2147483647
Max number of queued signals per process is	32
Max number of timers per process is	32
Supports asynchronous I/O is	TRUE
Supports File Synchronization is	TRUE
Supports memory mapped files is	TRUE

D E V I C E I N F O R M A T I O N

SUNW,Ultra-1 is a "Sun 501-2836"
openprom1 is a "Sun Open Boot PROM" device
options0 is a "PROM Settings"
aliases1 is a "PROM Device Aliases"
sbus0 is a "Sun SBus" system bus
audiocs0 is a "Crystal Semiconductor 4231" audio device
auxio is a "Auxiliary I/O"
flashprom1 is a "Sun Flash PROM"
eeprom1 is a "EEPROM" device
zs0 is a "Zilog 8530" serial device
zs1 is a "Zilog 8530" serial device
espdma is a "SCSI DMA" pseudo device
esp0 is a "Generic SCSI" SCSI controller
c0t0d0 (sd0) is a "SUN2.1G" 2.0 GB disk drive
ledma is a "LANCE Ethernet DMA" pseudo device
le0 is a "AMD Lance Am7990" 10-Mb Ethernet network interface
bpp is a "Sun Bidirectional Parallel Port" parallel device
cpu0 is a "Sun UltraSPARC" 143 MHz CPU

4. Folyamatok

- A folyamat (processzus) fogalma
- Folyamat ütemezés (scheduling)
- Folyamatokon végzett "műveletek"
- Folyamatok együttműködése, kooperációja
- Szálak (thread)
- Folyamatok közötti kommunikáció

• A folyamat (processzus) fogalma

- A folyamat (processzus): végrehajtás alatt álló program.

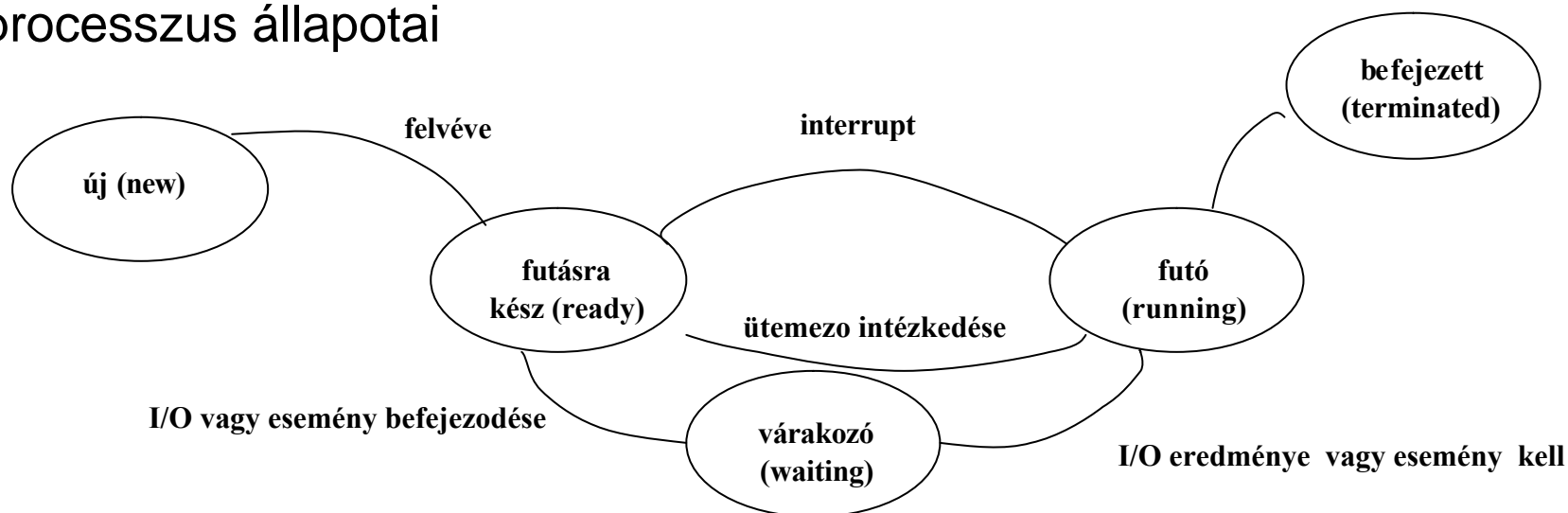
Betölthető program (jellemzői)

- passzív
- lemezen tárolt
- betöltési cím
- belépési pont
- ...

Processzus (jellemzői)

- aktív
- a memóriában van/volt
- program címszámláló értéke
- regiszterek értéke
- lokális/globális változók értéke
- verem állapota
- ... core dump ...

• A processzus állapotai



• Processzus - vezérlő blokk (Process Control Block – PCB)

- processzus állapot (process state) : new, ready, running, waiting, halted, sleeping, ...
- Program címszámláló (PC) értéke
- CPU regiszterek tartalma
- memória foglalási adatok
- account/user adatok
- I/O státusz információ (a folyamathoz rendelt I/O erőforrások, állományok listája)

• Processzus állapot információk (UNIX: ps, top; NT: Task manager)

TOP: last pid: 9099; load averages: 0.00, 0.00, 0.01 11:33:35
 29 processes: 28 sleeping, 1 on cpu
 CPU states: 99.6% idle, 0.2% user, 0.2% kernel, 0.0% iowait, 0.0% swap
 Memory: 128M real, 58M free, 120M swap free

PID	USERNAME	THR	PRI	NICE	SIZE	RES	STATE	TIME	CPU	COMMAND
9099	fazekas	1	58	0	2608K	1952K	cpu	0:00	0.42%	top
226	root	1	58	0	912K	672K	sleep	0:00	0.02%	utmpd
4081	root	1	58	0	1904K	1648K	sleep	0:00	0.02%	sshd
4084	fazekas	1	48	0	2600K	2160K	sleep	0:00	0.00%	tcsh
583	root	1	45	0	1760K	1024K	sleep	3:30	0.00%	sshd
1	root	1	58	0	680K	312K	sleep	0:02	0.00%	init
166	root	5	58	0	2720K	2144K	sleep	0:01	0.00%	automountd
8895	root	1	59	0	9208K	10M	sleep	0:01	0.00%	Xsun
239	root	5	22	0	2296K	1928K	sleep	0:00	0.00%	vold
212	root	3	22	0	1168K	848K	sleep	0:00	0.00%	powerd
170	root	8	35	0	3608K	1936K	sleep	0:00	0.00%	syslogd
592	root	1	38	0	1560K	1120K	sleep	0:00	0.00%	ttymon

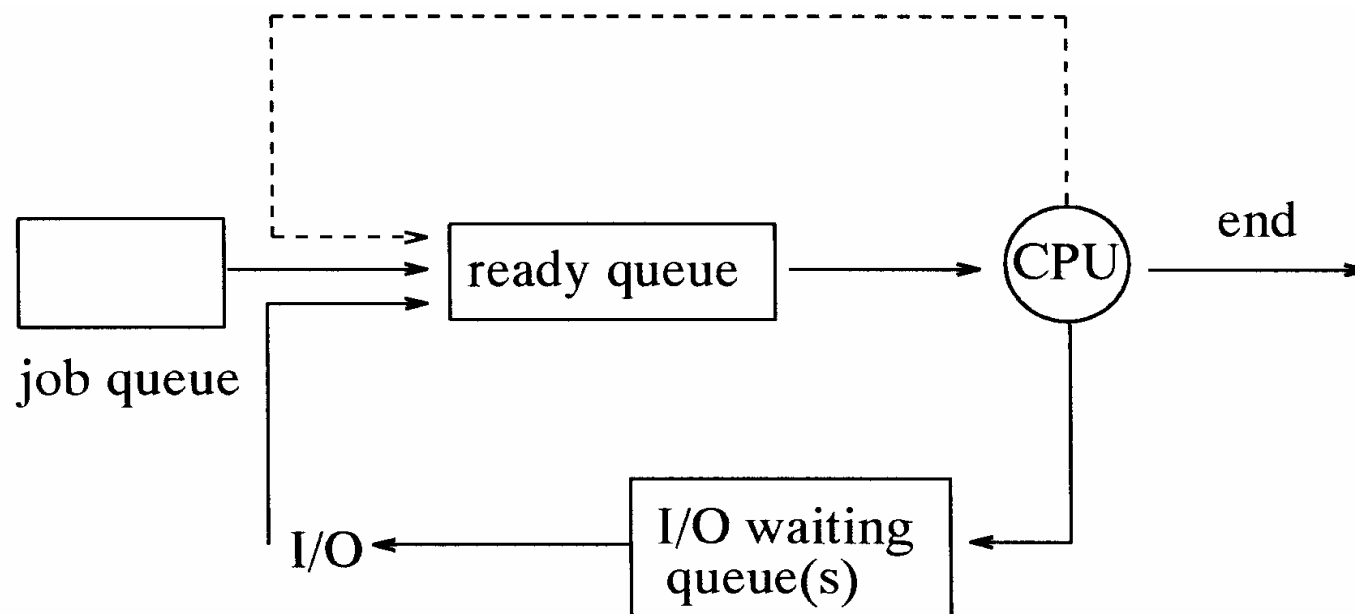
PS:

USER	PID	%CPU	%MEM	SZ	RSS	TT	S	START	TIME	COMMAND
------	-----	------	------	----	-----	----	---	-------	------	---------

fazekasg	513	1.1	2.012192	9848	?	S	15:00:12	0:38	/usr/openwin/bin/Xsun :0 -nobanner -auth /var/dt/A:0-
0s6gpT									
root	989	0.2	0.2	1184	1064	pts/6	O	08:41:39	0:00 ps -auxx
fazekasg	722	0.1	0.4	1992	1784	pts/6	R	15:39:28	0:00 /bin/tcsh
fazekasg	719	0.1	0.7	3840	3128	??	S	15:39:28	0:00 /usr/openwin/bin/cmdtool
fazekasg	984	0.1	0.7	4528	3424	?	S	08:40:53	0:00 /usr/openwin/bin/textedit
root	3	0.1	0.0	0	0	?	S	14:52:17	0:37 fsflush
fazekasg	744	0.1	0.7	3864	3176	??	S	15:57:25	0:00 /usr/openwin/bin/cmdtool
fazekasg	620	0.1	0.4	2472	2080	pts/2	S	15:01:22	0:00 olwm -syncpid 619
fazekasg	733	0.1	0.7	3848	3120	??	S	15:43:04	0:00 /usr/openwin/bin/cmdtool
fazekasg	636	0.0	1.0	5496	4896	pts/2	S	15:01:38	0:03 /usr/openwin/bin/filemgr -Wp 0 291 -Ws 592 439 -WP 81 833
+Wi									
root	0	0.0	0.0	0	0	?	T	14:52:16	0:00 sched
root	1	0.0	0.1	664	312	?	S	14:52:17	0:00 /etc/init -
root	2	0.0	0.0	0	0	?	S	14:52:17	0:00 pageout
root	119	0.0	0.3	1872	1240	?	S	14:53:04	0:00 /usr/sbin/rpcbind
root	121	0.0	0.3	1992	1296	?	S	14:53:04	0:00 /usr/sbin/keyserv
root	127	0.0	0.3	1920	1488	?	S	14:53:05	0:00 /usr/sbin/nis_cachemgr
root	149	0.0	0.4	1928	1712	?	S	14:53:18	0:00 /usr/sbin/inetd -s
root	154	0.0	0.4	2264	1824	?	S	14:53:18	0:00 /usr/lib/nfs/statd
root	156	0.0	0.3	1864	1528	?	S	14:53:18	0:00 /usr/lib/nfs/lockd
root	174	0.0	0.5	2912	2488	?	S	14:53:19	0:00 /usr/lib/autofs/automountd
root	178	0.0	0.4	3712	2032	?	S	14:53:20	0:00 /usr/sbin/syslogd
root	189	0.0	0.3	1808	1456	?	S	14:53:20	0:00 /usr/sbin/cron
root	201	0.0	0.5	2824	2512	?	S	14:53:21	0:00 /usr/sbin/nsd
root	211	0.0	0.2	2640	952	?	S	14:53:24	0:00 /usr/lib/lpsched
root	229	0.0	0.2	1152	840	?	S	14:53:27	0:00 /usr/lib/power/powerd
root	232	0.0	0.4	2104	1696	?	S	14:53:27	0:00 /usr/lib/sendmail -bd -qlh
root	244	0.0	0.2	888	712	?	S	14:53:28	0:00 /usr/lib/utmpd
root	259	0.0	0.4	2248	1936	?	S	14:53:29	0:01 /usr/sbin/vold
root	289	0.0	0.3	1840	1376	?	S	14:53:35	0:00 /usr/lib/snmp/snmpdx -y -c /etc/snmp/conf
root	298	0.0	0.4	2496	1704	?	S	14:53:37	0:00 /usr/lib/dmi/dmispd
root	299	0.0	0.5	3144	2304	?	S	14:53:38	0:00 /usr/lib/dmi/snmpXdmid -s pader

• Folyamat ütemezés (scheduling)

- Cél: mindig legyen legalább egy processzus, amelyik képes és kész a processzort lefoglalni.
- Processzus ütemezési sorok:
 - job queue (munka sor)
 - ready queue (készenléti sor)
 - device queue (berendezésre váró sor)
- Folyamat migráció az egyes sorok között:



- Folyamat ütemezők:

- Hosszútávú ütemező (*long term scheduler, job scheduler*)

- mi kerül a job queue-ba, lehet lassú, a multiprogramozás foka

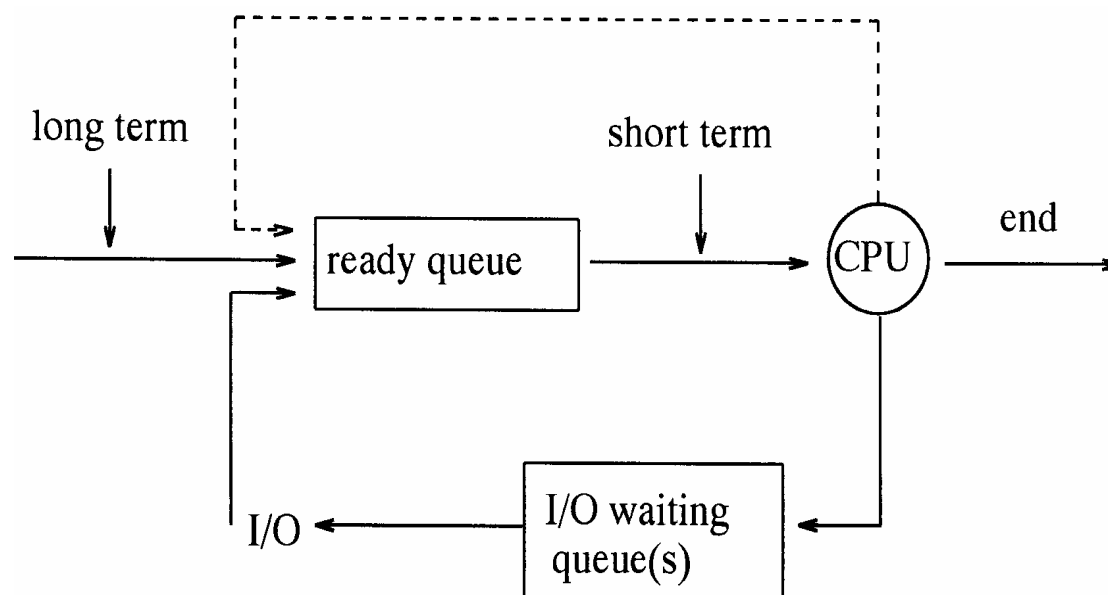
- Rövidtávú ütemező (*short term scheduler, CPU scheduler*)

- melyik folyamat kapja meg következő alkalommal a CPU-t, gyorsaság lényeges

- Szempontok:

- I/O igényes és CPU igényes folyamatok

- Context switch (process context a processzus továbbindításához szükséges összes információ rendszerezve, struktúrálva – kapcsolódó adatszerkezetek.)



- **Folyamatokon végzett "műveletek" (operációk)**
- **Processzus létrehozása**
 - (kezdeti betöltést kivéve processzust csak processzus hozhat létre!)
- **Mechanizmusa:** egy szülő (parent) folyamat létrehozhat gyermek (child) folyamatokat, majd a gyermekek további gyermekeket → fastruktúra
- **Erőforrás megosztás:**
 - Szülő és gyermek közösen használ minden erőforrást.
 - A gyermek a szülő erőforrásainak egy részét használhatja.
 - Nincs közös erőforráshasználat.
- **Végrehajtás:**
 - Szülő és gyermek konkurens módon fut. (UNIX: *parancs&*)
 - Szülő a gyermekre vár. (UNIX: *parancs*)
- **Címtér (address space)**
 - A gyermek a szülő duplikáltja.
 - A gyermek betölt egy programot önmaga helyett .
- **UNIX példák: `fork`, `execve`**

- **Processzus megszün(tet)ése (termination)**
- A folyamat végrehajtja az utolsó utasítását, majd megkéri az operációs rendszert, hogy törölje (exit). Ennek során
 - output adatok kerülnek át a gyermektől a szülőhöz (cf. **fork**),
 - a folyamat által használt erőforrásokat az operációs rendszer felszabadítja.
- A szülő folyamat felfüggesztheti a gyermek folyamatok végrehajtását (**abort**). Ennek lehetséges okai pl.:
 - A gyermek kimerítette a hozzárendelt erőforrásokat.
 - A gyermekhez rendelt taskra nincs többé szükség.
 - Maga a szülő is befejezi működését.
- Az operációs rendszer (legtöbb esetben) nem engedi meg, hogy egy gyermek tovább éljen, mint a szülő!
- Kaszkád termináció (A szülővel együtt elhal az összes gyermeke).
- Példa: UNIX –
 - gyermek: **exit** rendszerhívás segítségével jelzi, hogy nem kíván tovább működni.
 - szülő: **wait** rendszerhívással várakozhat egy gyerek befejeződésére.
 - az operációs rendszer a szülő befejezésével a gyerekeket is megszünteti!
- **Folyamatok együttműködése, kooperációja**

- Független folyamatok nem befolyásolnak más folyamatokat és nem befolyásolhatók más folyamatok által.
- Együttműködő folyamatok befolyásolhatnak más folyamatokat és befolyásolhatók lehetnek más folyamatok által.
- A kooperáció (kooperációt lehetővé tevő operációs környezet) több vonatkozásban előnyös lehet:
 - Információ megosztás
 - Számítási folyamatok felgyorsítása
 - Modularitás
 - Kényelem, kényelmi szempontok.
- Termelő-fogyasztó folyamatok (a kooperáló folyamatok egy paradigmája)
 - Termelő (producer) folyamat valamilyen adatot hoz létre.
 - Fogyasztó (consumer) folyamat az adatot fel(él)- használja.
 - A kooperációhoz valamilyen puffer ("raktár") szükséges.
 - Korlátlan ("végtelen") puffer \Leftrightarrow korlátos ("véges") puffer megoldás.

- Közös memória véges pufferrel megoldás

Megosztott adat	Termelő folyamat	Fogyasztó folyamat
var <i>n</i> ; type <i>item</i> = ... ; var <i>buffer</i> : array [0.. <i>n</i> -1] of <i>item</i> ; <i>in</i> , <i>out</i> : 0.. <i>n</i> -1; <i>in</i> :=0; <i>out</i> :=0;	repeat ... egy item létrehozása a <i>nextp</i> változóba ... while (<i>in</i> +1) mod <i>n</i> = <i>out</i> do <i>nop</i> ; <i>buffer</i> [<i>in</i>] := <i>nextp</i> ; <i>in</i> := (<i>in</i> +1) mod <i>n</i> ; until <i>false</i> ;	repeat while <i>in</i> = <i>out</i> do <i>nop</i> ; <i>nextc</i> := <i>buffer</i> [<i>out</i>]; <i>out</i> := (<i>out</i> +1) mod <i>n</i> ; ... a <i>nextc</i> változóban levő adat "felhasználása" ... until <i>false</i> ;

- A megoldás korrekt, de csak *n*-1 puffert tud feltölteni!

- Szálak (fonalak, könnyűsúlyú folyamatok, *thread*, *lightweight process*)
- A szál a CPU kiszolgálás egy alapegysége; jellemzői:
 - program címszámláló
 - regiszter készlet (tartalma)
 - verem tartalma
- Egy szál megosztva használja a vele egyenrangú (társ)- szálakkal a
 - kód szekcióját,
 - adat szekcióját,
 - az operációs rendszer által biztosított erőforrásait;
 - ezek együttesét task-nak nevezzük.
 - Egy hagyományos, vagy könnyűsúlyú processzus nem más, mint egy task egyetlen szállal.
- Előny: csökken a "context-switch" végrehajtására fordított idő!
 - Egy több szálat tartalmazó task esetén míg az egyik szál várakozik (blokkolt), a másik futhat.
 - Példák: fájl szerver, web - böngésző.

- A szálak olyan mechanizmust szolgáltatnak, amely lehetővé teszi a szekvenciális processzusoknak a rendszerhívások blokkolását, s közben a "párhuzamosság elérését".
- Megvalósítása:
 - Kernel által támogatott szálak (Mach és OS/2)
 - Felhasználói szintű szálak
 - a kernel szint fölött helyezkednek el;
 - nincs *system call*, csak *library call* (a fejlesztő rendszer/fordító oldja meg);
 - Andrew project (CMU)
 - vegyes (hibrid) megközelítés: kernel- és felhasználói szintű szálak
 - Solaris 2 (1992)
- A szálak állapotaik, kezelésük, egyéb tulajdonságaik (kommunikáció, kooperáció) alapján a processzushoz állnak közel.

• Processzusok közötti kommunikáció (IPC)

Az IPC olyan mechanizmust jelent, amely lehetővé teszi, hogy folyamatok egymással kommunikáljanak, és akcióikat összehangolják, szinkronizálják.

- Üzenő rendszer: a folyamatok úgy kommunikálnak, hogy nem rendelkeznek közösen használható memóriával.
- IPC két műveletet nyújt:
 - **send(message)**
 - az üzenetek lehetnek fix, vagy változó hosszúak
 - **receive(message)**
 - példák (UNIX rendszerhívások: `send`, `sendmsg`, `socket`, `recv`, `recvfrom`, ...)
- Ha a P és Q folyamatok kommunikálni szeretnének, akkor szükségük van egy kommunikációs vonalra (communication link)
- A kommunikációs vonal implementációs kérdései
 - Fizikai implementáció: megosztott memória, hardver busz, hálózat, ...
 - Logikai implementáció:
 - Hogyan épül fel a link?
 - Tartozhat-e egynél több folyamathoz?
 - Két folyamat között hány élő link lehet? ... üzenet mérete, mozgás iránya, stb

- **Direkt kommunikáció**

- **send**(P, message)
 - küldj egy üzenetet P-nek (utasítás Q-ban)
- **receive**(Q, message)
 - fogadj egy üzenetet Q-tól (utasítás P-ben)
- A kommunikációs vonal ebben az esetben automatikusan épül fel a két folyamat között (PID ismerete szükséges!)
- A vonal pontosan két folyamat között létezik.
- Minden egyes folyamatpár között pontosan egy link létezik
- Egy, vagy többirányú is lehet.
- Termelő-fogyasztó példa
- Szimmetrikus/asszimmetrikus címezés

- Indirekt kommunikáció

- **send**(A, message)

- küldj egy üzenetet az A Mail-boxba (utasítás Q-ban)

- **receive**(A, message)

- olvass ki egy üzenetet az A Mail-boxból (utasítás P-ben)

- A kommunikációs vonal abban az esetben épül fel a két folyamat között ha közösen használhatják az A Mail-boxot (PID ismerete nem szükséges!)
 - A vonal több folyamat között létezik (mindenki, aki A-hoz hozzáférhet!).
 - Minden egyes folyamatpár között több link is létezik, létezhet (Mail-box-függő)
 - Egy, vagy többirányú is lehet.
 - “Ki kapja az üzenetet?” - probléma.

- Pufferelés (a link által egyidőben befogadott üzenetek száma)

- Zéró kapacitás (0 üzenetet tárol a link; szinkronizáció kell, “rendezvous”)
 - Korlátozott kapacitás (n üzenet lehet a linken)
 - Korlátlan kapacitás
 - Más: delay, reply, RPC (remote procedure call)

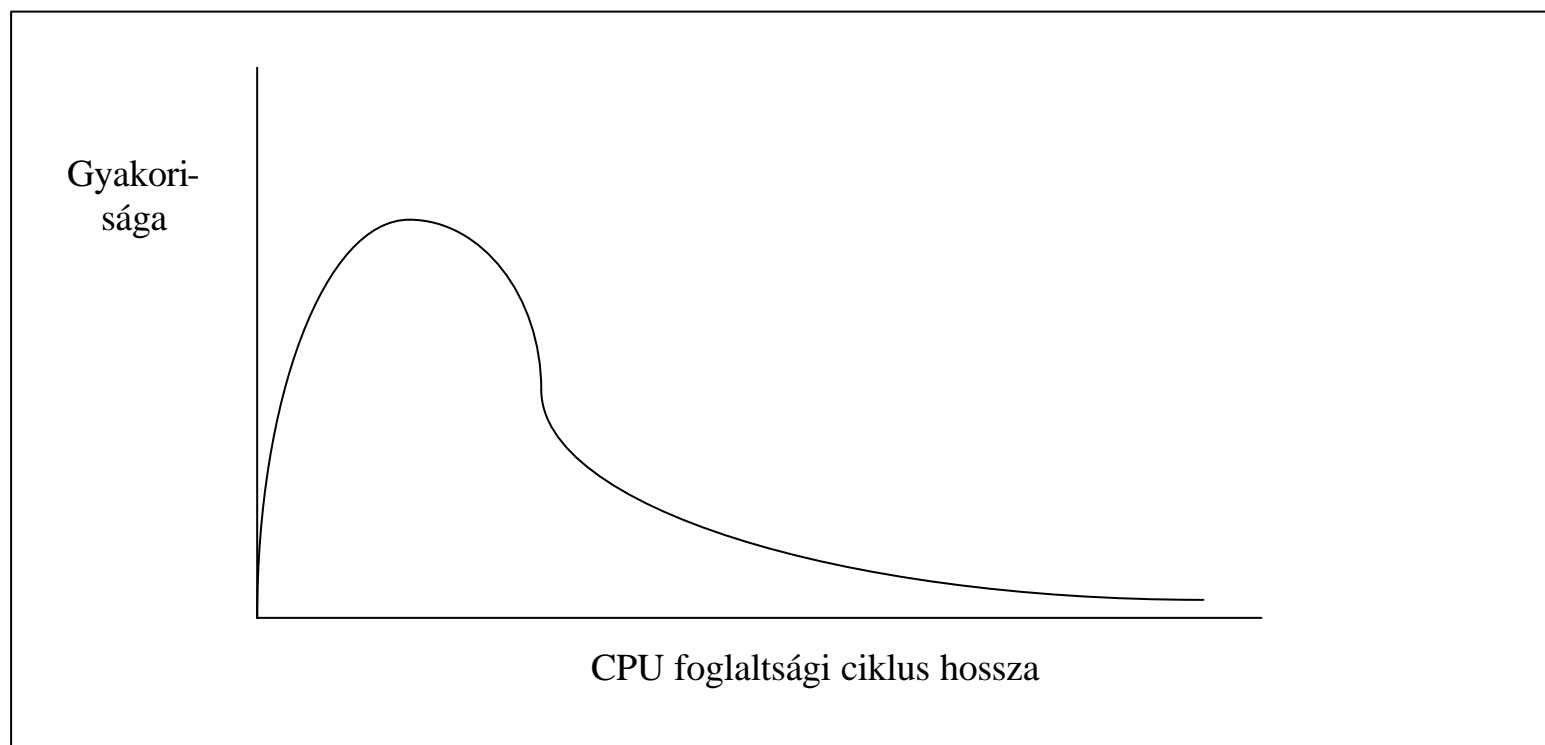
- Kivétel-kezelés, kivételes helyzetek
 - processzus felfüggesztés
 - elveszett üzenetek
 - hibás (scrambled) üzenetek

5. CPU ütemezés

- Alapfogalmak
- Ütemezési feltételek (kritériumok)
- Ütemezési algoritmusok
- Több-processzoros eset
- Valós idejű (real time) ütemezés
- Algoritmus kiértékelés

•Alapfogalmak

- A multiprogramozás célja: a CPU foglaltság (kihasználás) hatásfokának növelése
- A multiprogramozás lényege: egyidőben több folyamat is az operatív tárban helyezkedik el, készen állva a CPU kiszolgálására. Ha egy folyamatnak várnia kell (PI. I/O-ra), a CPU-t egy másik folyamat kapja meg.
- CPU Burst cycle – I/O Burst Cycle (CPU foglaltsági ciklus – I/O ciklus)
- A CPU ciklusok gyakorisága:



- Rövid távú (*short term*) ütemezőről van szó
- Rövid távú ütemezési döntéshelyzet áll elő, ha egy folyamat
 1. futó állapotból várakozó állapotba kerül
 2. futó állapotból készenléti állapotba kerül
 3. várakozó állapotból készenléti állapotba kerül
 4. megáll
- Az 1. és 4. esetben az ütemezés *nem preemptív* (nem beavatkozó: a processzus maga mond le a CPU-ról) – (MS Win)
- Az 2. és 3. esetben az ütemezés *preemptív* (beavatkozó: elveszik tőle a CPU-t)
- A Diszpécser (dispatcher)
 - A diszpécser adja át a vezérlést az ütemező által kiválasztott folyamatnak. Ez magában foglalja:
 - a kontextus módosítását
 - user módba kapcsolást
 - a megfelelő című utasításra ugrást (PC/IP beállítást)
- Diszpécser latencia : egy processzus megállításhoz és egy másik elindításához szükséges idő

• Ütemezési feltételek (kritériumok)

- **CPU kiszolgálás** (CPU utilization) – a CPU az idő minél nagyobb részében legyen foglalt.
- **Átbocsátó képesség** (throughput) – egységnyi idő alatt befejeződő processzusok száma.
- **Végrehajtási** (turnarund, fordulási) **idő** – $= \sum$ várakozás a memóriába kerülésre + készenléti sorban töltött idő + CPU-n töltött idő + I/O-val töltött idő.
- **Várakozási idő** (waiting time) – készenléti sorban (ready queue) töltött idő
- **Válaszidő** (response time) – a “kérés benyújtásától” az első válasz megjelenéséig (nem output!) eltelt idő. (Interaktív rendszerekben a turnaround nem jó jellemző.)
- **Feladatok:** “tisztességes” kiszolgálás, minimum, maximum, átlag optimalizálása, szórás optimalizálása, jósolhatóság.

- **Ütemezési algoritmusok**
- **Igénybejelentési sorrend szerinti kiszolgálás** (first-come, first-served = FCFS)
 - az átlagos várakozási idő nagy szórása, konvoj effektus
- **A rövidebb igény először** (shortest job first, shortest job next, SJF, SJN) kiszolgálás elve.
 - Elméletileg minimalizálja az átlagos várakozási időt.
 - Probléma: nem tudjuk, hogy a sorbanállók közül melyik a “rövidebb”.
 - Printer spooling.
 - Predikció: a korábbi foglaltsági periódusok hosszának felhasználásával pl. exponenciális átlagolással (aging - öregedési algoritmus)

Jelölje t_n az n -edik CPU foglaltsági ciklus hosszát, t_{n+1} a következőnek jóslott foglaltsági periódus hosszát.

Definiáljuk a következő foglaltság várható hosszát: Legyen $1 \geq \alpha \geq 0$ és

$$t_{n+1} = \alpha t_n + (1-\alpha) t_n$$

- Lehet preemptív, vagy non-preemptív.

- **Prioritási sorrend szerinti kiszolgálás**

- Prioritás: a processzushoz rendelt egész érték. A kisebb érték nagyobb prioritást jelent.
- Prioritási függvény
 - Az SJF, ill. FCFS is prioritásos kiszolgálásnak tekinthető!
- Belső és külső prioritás.
- Preemptív, non-preemptív prioritási ütemezők.
- Problémák: végtelen indefinit blokkolódás = éhezés, éhhalál.
 - Példa: MIT 1973 IBM 7094: 1967 óta várt egy processzus a CPU-ra!!
- Megoldás: aging (unix példa)

- **Round robin (RR, körleosztásos, körbejáró ütemezés)**

- Minden processzus sorban q ideig ($q=10-100$ millisec.) használhatja a CPU-t
- n processzus esetén a maximális várakozási idő $(n-1)/q$
- q nagy: \Rightarrow FCFS !
- q kicsi: \Rightarrow kontextus váltási költség relatíve megnő!

6. Folyamat szinkronizáció

- Háttér
- A kritikus szakasz probléma
- Szinkronizációs hardver
- Szemaforok
- Klasszikus szinkronizációs problémák
- Kritikus régiók
- Monitorok

•Háttér

- A termelő-fogyasztó folyamat egy módosított változata (counter!)

Megosztott adat	Termelő folyamat	Fogyasztó folyamat
var <i>n</i> ; type <i>item</i> = ... ; var <i>buffer</i> : array [0.. <i>n</i> -1] of <i>item</i> ; <i>in</i> , <i>out</i> , <i>counter</i> : 0.. <i>n</i> -1; <i>in</i> :=0; <i>out</i> :=0; <i>counter</i> =0;	repeat ... egy item létrehozása a <i>nextp</i> változóba ... while <i>counter</i> = <i>n</i> do <i>nop</i> ; <i>buffer</i> [<i>in</i>] := <i>nextp</i> ; <i>in</i> := (<i>in</i> +1) mod <i>n</i> ; <i>counter</i> = <i>counter</i> +1; until <i>false</i> ;	repeat while <i>counter</i> = 0 do <i>nop</i> ; <i>nextc</i> := <i>buffer</i> [<i>out</i>]; <i>out</i> := (<i>out</i> +1) mod <i>n</i> ; <i>counter</i> = <i>counter</i> -1; ... a <i>nextc</i> változóban levő adat "felhasználása" ... until <i>false</i> ;

- A kód külön-külön korrekt, de konkurens módon végrehajtva nem az!

- A $counter := counter \pm 1$ utasítás assembly szintű implementálása gépi utasításokkal:

$reg1 := counter$	$reg2 := counter$
$reg1 := reg1 + 1$	$reg2 := reg2 - 1$
$counter := reg1$	$counter := reg2$

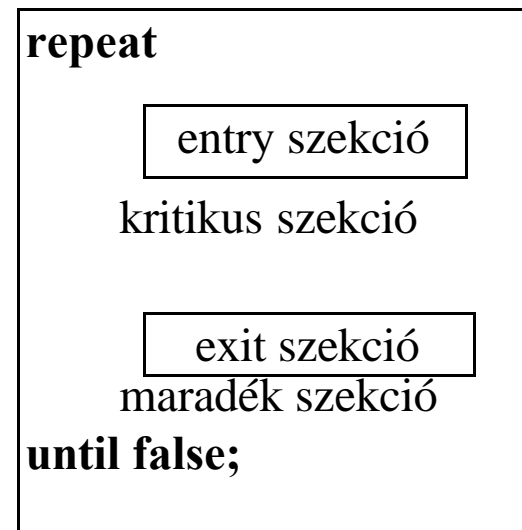
- Legyen a $counter$ értéke pl. 5! Akkor egy konkurens hozzáférés-pár után (Megszakítás minden gépi utasítás után bekövetkezhet!):

L1	Termelő	$reg1 := counter$	$reg1 := 5$
L2	Termelő	$reg1 := reg1 + 1$	$reg1 := 6$
L3	Fogyasztó	$reg2 := counter$	$reg2 := 5$
L4	Fogyasztó	$reg2 := reg2 - 1$	$reg2 := 4$
L5	Termelő	$counter := reg1$	$counter := 6$
L6	Fogyasztó	$counter := reg2$	$counter := 4$

Azaz a $counter$ értéke 4 (és nem 5!).

• A kritikus szakasz probléma

- n folyamat verseng egy bizonyos (közös) adat használatáért
- Mindegyik folyamatnak van egy kódszegmense, ahol ezt a bizonyos adatot eléri (olvassa/írja): ez az ún. *kritikus szakasz*.
- Probléma: hogyan biztosítható, hogy amíg egy processzus a kritikus szakaszát hajtja végre, addig egyetlen más processzus se léphessen be a saját kritikus szakaszába.
- Az i -edik processzus (P_i) szerkezete:



• Követelmények:

- Kölcsönös kizárás
 - nem lehet egynél több processzus a kritikus szakaszában
- Progresszió
 - a kiválasztás (futásra) nem késleltethető határozatlan ideig
- Korlátozott várakozás
 - minden igény kiszolgálása megkezdődik korlátozott számú kritikus szakasz végrehajtása után

• Algoritmusok két processzus esetére

I. algoritmus	II. algoritmus	III. algoritmus
<ul style="list-style-type: none"> • közös változók var <i>turn</i> : (0..1); {kezdeti érték: <i>turn</i> =0; ha <i>turn</i> = <i>i</i> \Rightarrow P_i beléphet a kritikus szekciójába}	<ul style="list-style-type: none"> • közös változók var <i>flag</i> : array [0..1] of <i>boolean</i> ; {kezdeti érték: <i>flag</i> [<i>i</i>]= <i>false</i> ; ha <i>flag</i> [<i>i</i>]= <i>true</i> \Rightarrow P_i kész be- lépni a kritikus szekciójába}	<ul style="list-style-type: none"> • közös változók • I. és II. kombinációja
<ul style="list-style-type: none"> • Processzus P_i repeat <div> while <i>turn</i> \neq <i>i</i> do <i>no-op</i>; </div> <div>kritikus szekció</div> <div> <i>turn</i> = <i>j</i>; </div> <div>maradék szekció</div> until false; 	<ul style="list-style-type: none"> • Processzus P_i repeat <div> <i>flag</i>[<i>i</i>] := <i>true</i>; while <i>flag</i>[<i>j</i>] do <i>no-op</i>; </div> <div>kritikus szekció</div> <div> <i>flag</i>[<i>i</i>] := <i>false</i>; </div> <div>maradék szekció</div> until false; 	<ul style="list-style-type: none"> • Processzus P_i repeat <div> <i>flag</i>[<i>i</i>] := <i>true</i>; <i>turn</i> := <i>j</i>; while <i>flag</i>[<i>j</i>] and <i>turn</i>=<i>j</i> do <i>no-op</i>; </div> <div>kritikus szekció</div> <div> <i>flag</i>[<i>i</i>] := <i>false</i>; </div> <div>maradék szekció</div> until false;
Kölcsönös kizárás van, de progresszió nincs!	Progresszió van, de kölcsönös kizárás nincs!	Mindhárom feltételt kielégíti!

- Bakery algoritmus (kritikus szakasz n processzusra)
- Minden processzus kap egy sorszámot, mielőtt belép a kritikus szekciójába (ticket). A legkisebb sorszámú processzus hajthatja végre a kritikus szekcióját.
- Ha P_i és P_j sorszáma megegyezik, akkor a kisebb indexű jogosult a kritikus szekciójába belépni.
- A sorszámozó rendszer mindig monoton növekvő sorszámokat generál.
 - Pl.: 1,2,2,3,3,3,4,4, ...
- Jelölés: “<” jelentse a (ticket #, process ID) párra definiált lexikografikus rendezést.

Bakery algoritmus

- közös változók

var *choosing*: **array**[0..*n*-1] **of** *boolean*;

number: **array**[0..*n*-1] **of** *integer*;

{kezdeti érték: *i* = 0, 1, ... , *n*-1 -re *choosing*[*i*] := false, *number*[*i*] := 0.}

- **Processzus P_i**

repeat

choosing[*i*] := true;

number[*i*] := max(*number*[0], ... , *number*[*n*-1]) + 1;

choosing[*i*] := false;

for *j* := 0 **to** *n*-1 **do**

begin

while *choosing*[*j*] **do** no-op;

while *number*[*j*] ¹ 0 **and** (*number*[*j*], *i*) < (*number*[*i*], *i*) **do** no-op;

 end;

 kritikus szekció

number[*i*] := 0;

 maradék szekció

until false;

• Szinkronizációs hardver

- A **test-and-set (TS)** gépi utasítás (bizonyos) architektúrákban egyszerre képes egy memória szó tartalmát lekérdezni (+/0/- ?) és ugyanakkor a szóba egy új értéket beírni.
- Felhasználása: A közös adat elérésének ténye más processzusok számára érzékelhetővé tehető!

• Szemaforok (Dijkstra, 1965)

- Aktív várakozás (busy waiting) problémája
- Szemafor (S)
 - integer változó
 - csupán az alábbi két elemi (atomikus) művelet hajtható rajta végre

wait(S): $S := S - 1;$
 if $S < 0$ **then** *block(S)*

signal(S): $S := S + 1;$
 if $S \neq 0$ **then** *wakeup(S)*

- *block(S)* – felfüggeszti a hívó processzus végrehajtását
 - a processzust hozzáadja az S szemaforon alvó processzusok sorához
- *wakeup(S)* – reaktivál pontosan egy, korábban *block(S)* hívást kezdeményező folyamatot
 - az S szemaforon alvó processzusok sorából egyet felébreszt

- Deadlock – two or more processes are waiting indefinitely for an event that can be caused by only one of the waiting processes.

Let S and Q be two semaphores initialized to 1

P_0	P_1
$wait(S)$	$wait(Q)$
$wait(Q)$	$wait(S)$
.	.
.	.
.	.
$signal(S)$	$signal(Q)$
$signal(Q)$	$signal(S)$

- Starvation – indefinite blocking

A process may never be removed from the semaphore queue in which it is suspended.

- **Klasszikus szinkronizációs problémák**
- **A korlátos puffer probléma**
 - a modellezéshez használt szemaforok: *mutex*, *empty*, *full*
 - *empty* kezdőértéke: n
 - *full* kezdőértéke: 0
 - számláló (*counting*) szemaforok.
- **Az olvasók és írók (readers and writers) probléma**
 - Egy adatot, állományt több processzus megosztva, párhuzamosan használ, egyesek csak olvassák, mások csak írják. Hogyan biztosítható az adatok konzisztenciája?
 - Stratégiák:
 1. Minden olvasó azonnal hozzáférhet az adatokhoz, hacsak egy író nem kapott már engedélyt az írásra.
 2. Egy író azonnal írhat, ha erre kész és más írók éppen nem írnak.
 - Mindkét stratégia éhezéshez (starvation) vezethet, de van megoldás.

Bounded-Buffer Problem

- Shared data

```
type item = ...  
var buffer = ...  
    full, empty, mutex: semaphore;  
    nextp, nextc: item;  
    full := 0; empty := n ; mutex := 1;
```

- Producer process

```
    repeat  
        ...  
        produce an item in nextp  
        ...  
        wait(empty);  
        wait(mutex);  
        ...  
        add nextp to buffer  
        ...  
        signal(mutex);  
        signal(full);  
    until false;
```

- Consumer process

repeat

wait(full);

wait(mutex);

...

remove an item from *buffer* to *nextc*

...

signal(mutex);

signal(empty);

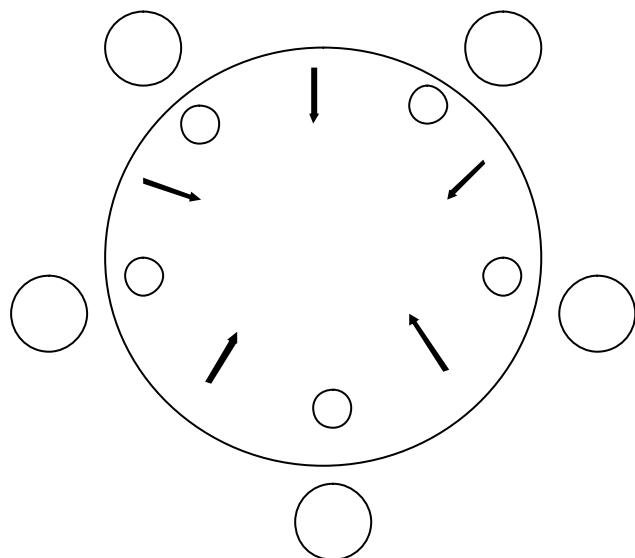
...

consume the item in *nextc*

...

until *false*;

• A vacsorázó filozófusok (dining philosophers) problémája



- Egy kör alakú asztal mellett öt filozófus ül, mindegyik előtt van egy tányér rizs és a szomszédos tányérok között egy-egy evő-pálcika.
- Evéshez a filozófus a saját tányérja melletti két evőeszközt használ-hatja úgy, hogy ezeket egymás után kézbe veszi.
- Ha befejezte az étkezést, vissza-teszi az eszközöket, és gondolkodni kezd.
- Majd újra megéhezik, stb.
- Példa számos konkurencia kontroll problémára.

• Kritikus régiók

- Magas szintű szinkronizációs eszköz.
- Egy megosztott változó v , amelynek típusa T , deklarációja

var v : shared T ;

- A v változó csak az alábbi alakú utasításokon keresztül érhető el:

region v when B do S ;

ahol B egy logikai kifejezés.

- Amíg az S utasítás végrehajtás alatt áll, másik processzus nem érheti el a v változót.
- Ha a processzus megpróbálja végrehajtani a region utasítást, a B logikai kifejezés kiértékelődik. Ha B igaz, az S utasítás végrehajtódik. Ha hamis, akkor a processzus végrehajtása késleltetődik addig, amíg a kifejezés igaz nem lesz és egyetlen más processzus sincsen a v -hez kapcsolt régióban.
- Példa: korlátos puffer problémája
- Implementáció: pl. szemaforokkal

- Példa: A korlátos puffer probléma egy lehetséges megoldása.

- Megosztott puffer:

```
var buffer: shared record
    pool: array [0..n-1] of item;
    count,in,out: integer;
end;
```

- *nextc* az osztott pufferbe:

```
region buffer when count < n
do begin
    pool[in] := nextp;
    in := in+1 mod n;
    count := count + 1;
end;
```

- Fogasztó kiveszi *nextc*-t

```
region buffer when count > 0
do begin
    nextc := pool[out];
    out := out+1 mod n;
    count := count - 1;
end;
```

Monitorok (Hoare, 1974; Hansen, 1975)

- magas szintű szinkronizációs eszközök (szinkronizációs primitív), amelyek lehetővé teszik egy absztrakt adattípus biztonságos megosztását konkurens processzusok között.
- formálisan: a monitor eljárások, változók és adatszerkezetek együttese, amelyek egy speciális csomagba vannak integrálva. A processzusok hívhatják a monitorban levő eljárásokat, de nem érhetik el közvetlenül annak belső adatszerkezetét.
- Speciális típus: **condition**
- Speciális műveletek: **x.wait**, **x.signal**
- A monitor azt kívánja biztosítani, hogy egy időben csak egy processzus legyen aktív (rajta).
- Megvalósítás: pl. szemaforokkal.

- Példa: A dining philosophers probléma egy lehetséges megoldása.

```
type dining-philosophers = monitor
  var state : array [0..4] of (thinking, hungry, eating);
  var self : array [0..4] of condition;

  procedure entry pickup (i: 0..4);
  begin
    state[i] := hungry;
    test (i);
    if state[i] ≠ eating then self[i].wait;
  end;

  procedure entry putdown (i: 0..4);
  begin
    state[i] := thinking;
    test (i+4 mod 5);
    test (i+1 mod 5);
  end;

  procedure test (k: 0..4);
  begin
    if state[k+4 mod 5] ≠ eating
      and state[k] = hungry
      and state[k+1 mod 5] ≠ eating
    then begin
      state[k] := eating;
      self[k].signal;
    end;
  end;

begin
  for i := 0 to 4
  do state[i] := thinking;
end.
```

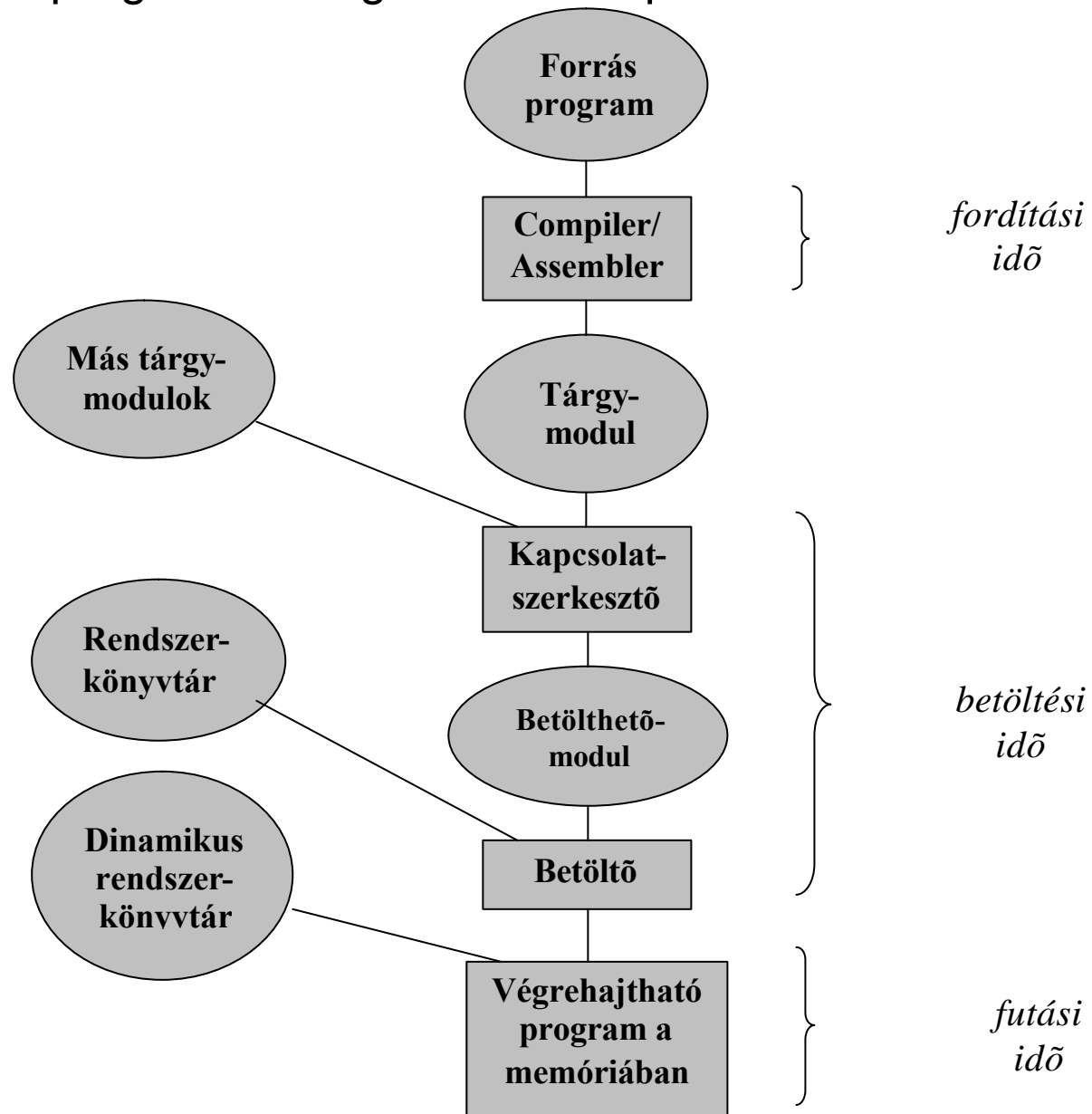
8. Memória management

- Háttér
- Logikai és fizikai címtér
- Swapping
- Folytonos allokálás
- Lapozás
- Szegmentáció
- Szegmentáció lapozással

•Háttér

- Az számítógép (processzor) kapacitásának jobb kihasználása megköveteli, hogy egyszerre több processzus osztozzon a memórián (shared memory).
- Egy program alapvetően valamilyen (bináris végrehajtható) fájl formában helyezkedik el a háttértárban. Végrehajtáshoz be kell tölteni a memóriába.
 - Végrehajtás közben – a memória kezelési stratégiától függően – többször mozoghat a memória és háttértár között.
- *Input queue* – a végrehajtásra kijelölt és evégett sorban álló programok együttese.
- A programkódhoz és a program változókhöz valamikor memória címeket kell rendelni (*address binding*). (Ez történhet a betöltés előtt, közben, vagy akár utána is.)
 - Fordítási időben történő tárfoglalás (címkapcsolás).
 - Betöltési (szerkesztési) időben történő tárfoglalás (címkapcsolás).
 - Futási időben történő tárfoglalás (címkapcsolás).

• Egy felhasználói program feldolgozásának lépései



• Dinamikus betöltés

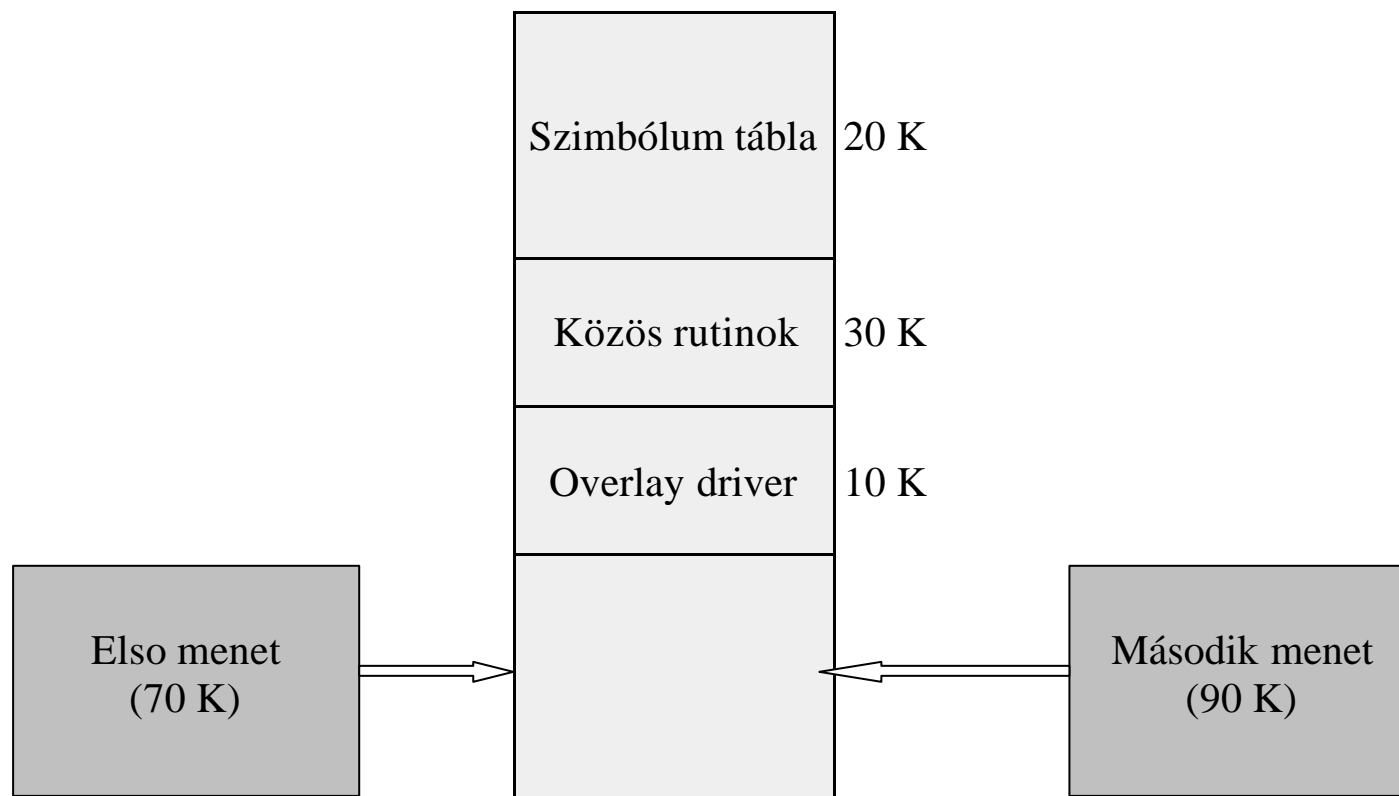
- Egy szubrutin nem töltődik be, amíg meg nem hívódik. Minden szubrutin a lemezen található áthelyezhető bináris formában.
- A hívó rutin először tisztázza, hogy a hívott benn van-e a memóriában. Ha nincs, akkor aktivizálódik az áthelyező betöltő, és a betöltés után a program címhivatkozásai (címtáblázat, címkonstansok) módosulnak.
- A nem szükséges rutinok soha nem töltődnek be!
- Nincs szükség speciális operációs rendszer támogatásra (a futtató rendszer - *run time system* saját hatáskörén belül megoldja).

• Dinamikus szerkesztés

- Statikus szerkesztés: a nyelvi könyvtárak úgy kezelődnek, mint bármely más (felhasználói) object modul. Probléma: a gyakran használt rutinok sok-sok végrehajtható program kódjával együtt letárolódnak. (lemez-pazarlás!)
- Dinamikus szerkesztésnél nemcsak az (áthelyező) betöltés, hanem a (szimbolikus) kapcsolat szerkesztés is kitolódik a futási időre. Egy kisméretű kód (*stub*=csonk) helyettesíti a szükséges rutint a végrehajtható program kódjában, amely segítségével a szükséges rutin a memóriában (ha az memória rezidens), vagy a lemezes könyvtárban lokalizálható. A lokalizálás után (következő alkalommal) a rutin már direkt módon hajtódik végre (nincs újra töltés!).
- További előnyök: könyvtár módosítások, verziók, *bug fixes*, *patches*, *service pack*, *shared library*.
- Operációs rendszer támogatás: védett területre betöltött rutinok elérése.

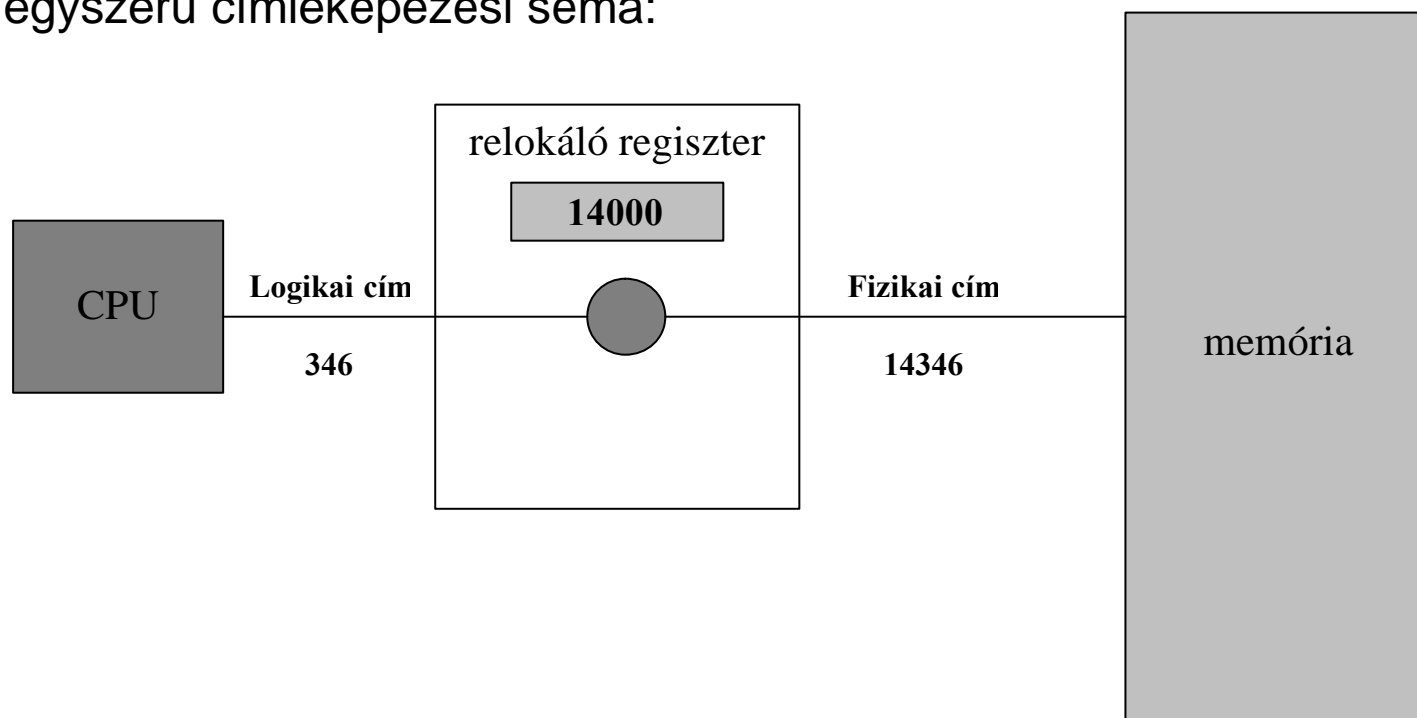
• Overlay

- A felhasználói program logikájába (szubrutin struktúrájába) beépített dinamika.
- Alapötlet: a teljes programnak (kód és adat) csak az a része legyen benn az operatív memóriában, amelyre ténylegesen szükség van.
- Példa: többmenetes fordítók, assemblerek.



• Logikai és fizikai címtér

- Logikai cím = a CPU által generált cím (virtuális cím).
- Fizikai cím = a *Memory Management Unit* (MMU) által generált cím (reális cím).
- A fordítási és betöltési időben csak a logikai címtér (címhozzárendelés) elérhető!
- A logikai címet a MMU képezi le fizikai címmé.
- Egy egyszerű címleképezési séma:

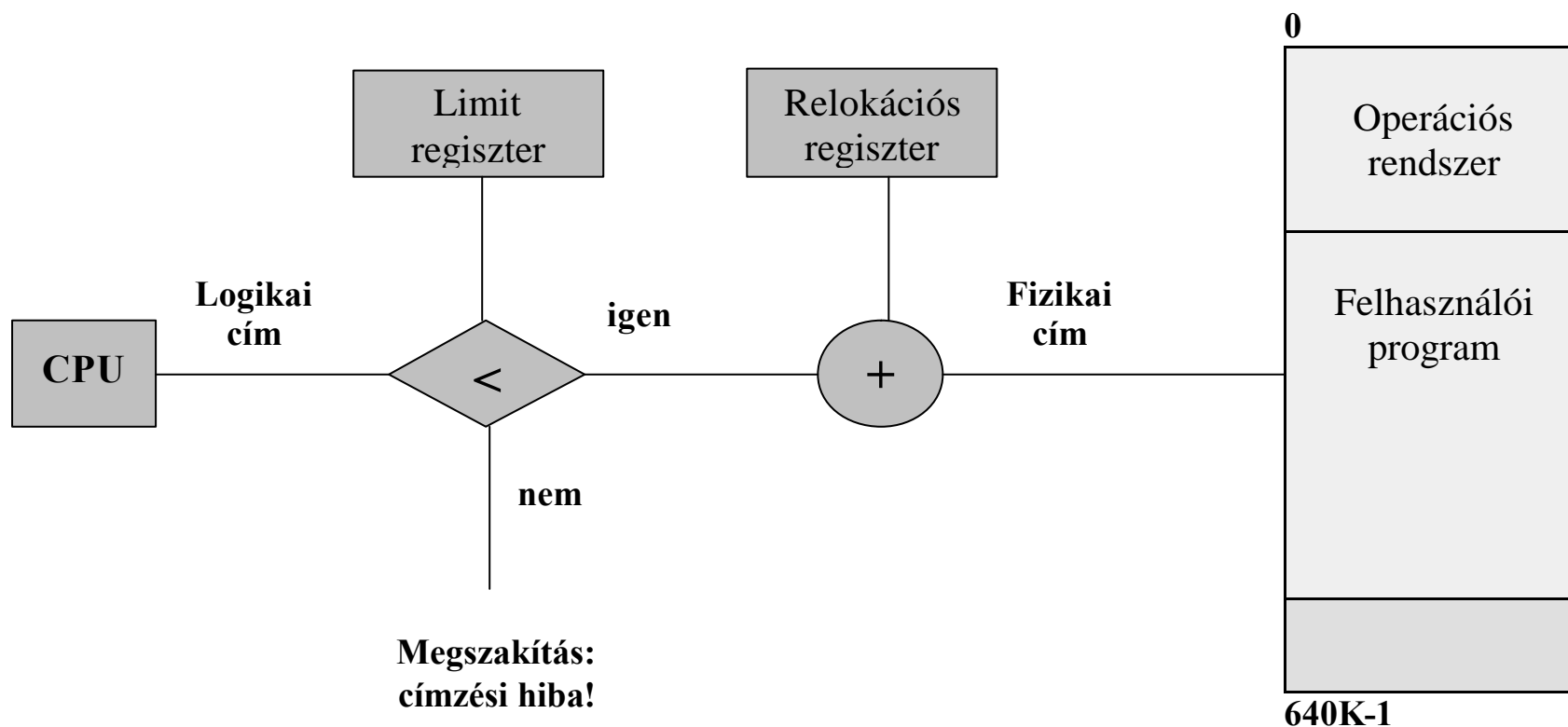


• Swapping

- swap = csere (egy futó processzus kódjának és adatainak "lecserélése" egy háttértárban tároltra).
- Háttértár: nagy, összefüggő, gyorsan elérhető lemezterület, amely elég nagy ahhoz, hogy minden futó program memóriabeli képét (core image) tárolja.
- Roll out, roll in: swapping stratégia változat: az alacsonyabb prioritású folyamatok kicserélődnek a magasabb prioritásúakra.
- A swap idő nagyrészt *adatátviteli idő(!)*, arányos a processzus által lefoglalt operatív memória méretével.
- A swapping egyes verziói megtalálhatók a UNIX-ban (automatikus) és az MS Windows-ban is (kézi vezérelt?).
- Round Robin példa: időosztás és átviteli idő.
- Más problémák: függő I/O.

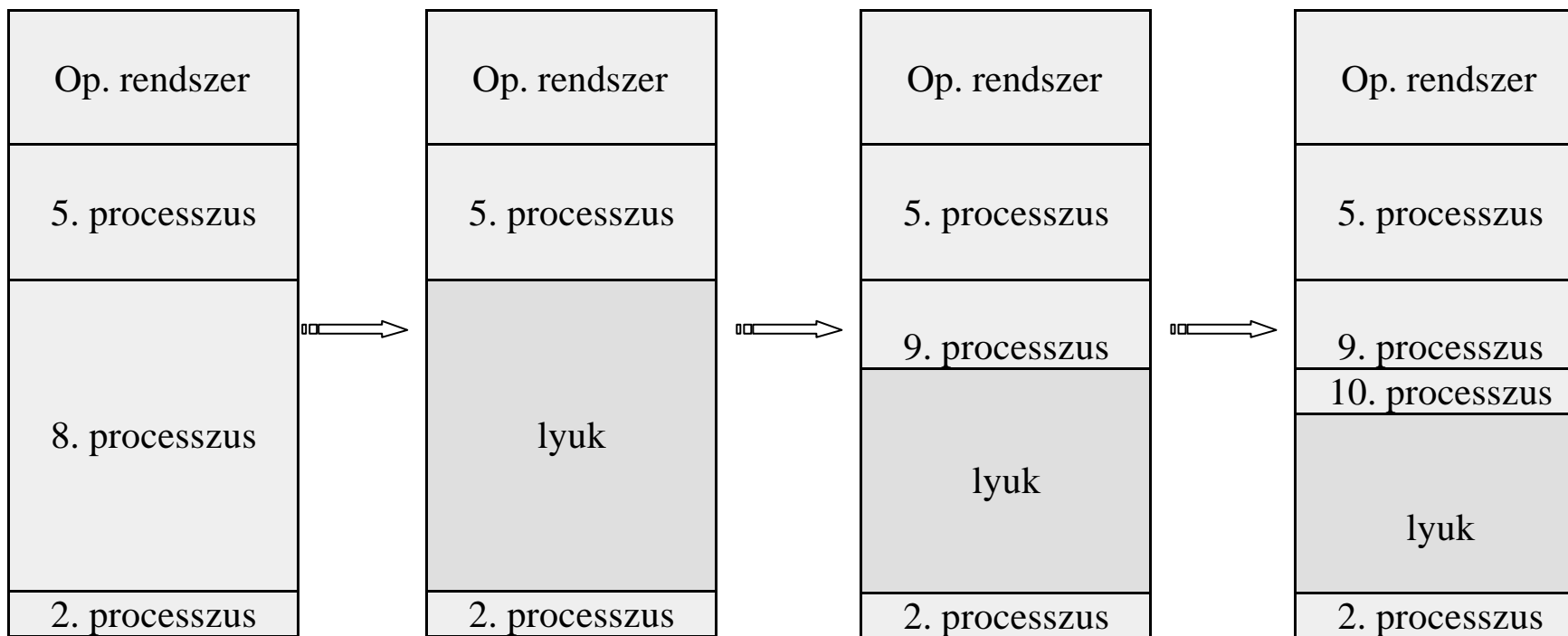
• Folytonos tár-allokálás

- Az operatív memóriát egyetlen folytonos tömbnek tekinthetjük.
 - A memória "rendszer"- és "felhasználói program" részekre bomlik.
 - A rendszer résznek tartalmaznia kell a memóriába beágyazott megszakítási vektort, az I/O kapukat (portok).
- Egyszerű partíciók allokálás



• Multipartíciós allokálás

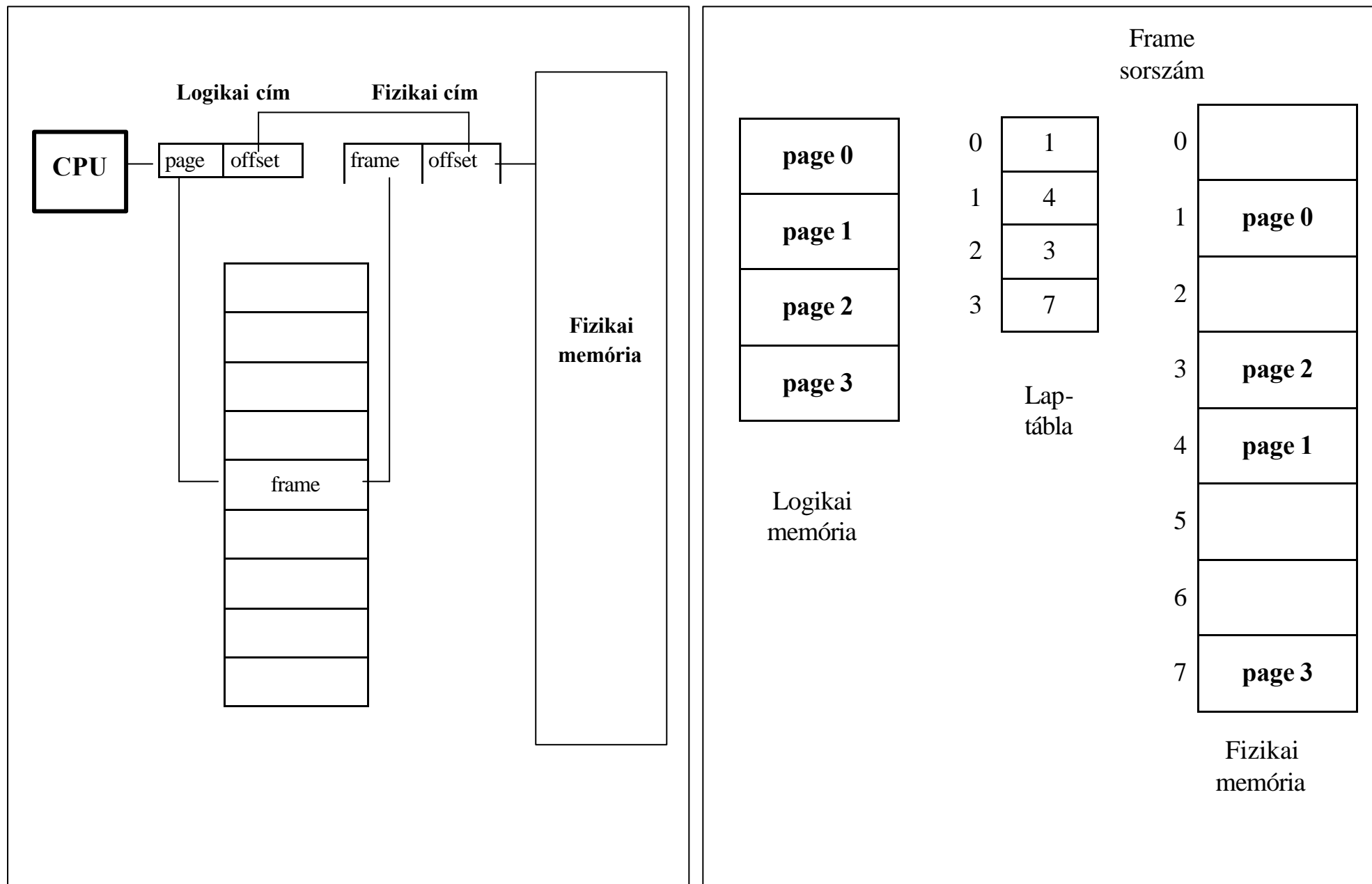
- A felhasználói programok számára elérhető terület részekre (partíciókra) bomlik. Egy partíció egy felhasználói program (processzus) befogadására alkalmas.
- Fix, változó számú és hosszúságú partíciók, prioritás.
- Lyuk (*hole*): két foglalt partíció közötti szabad memória terület (blokk). A lyukak mérete változó lehet.
- Ha egy processzust létre kell hozni, akkor ehhez az operációs rendszernek egy megfelelően nagy méretű lyukat kell kiválasztania.
- Példa:



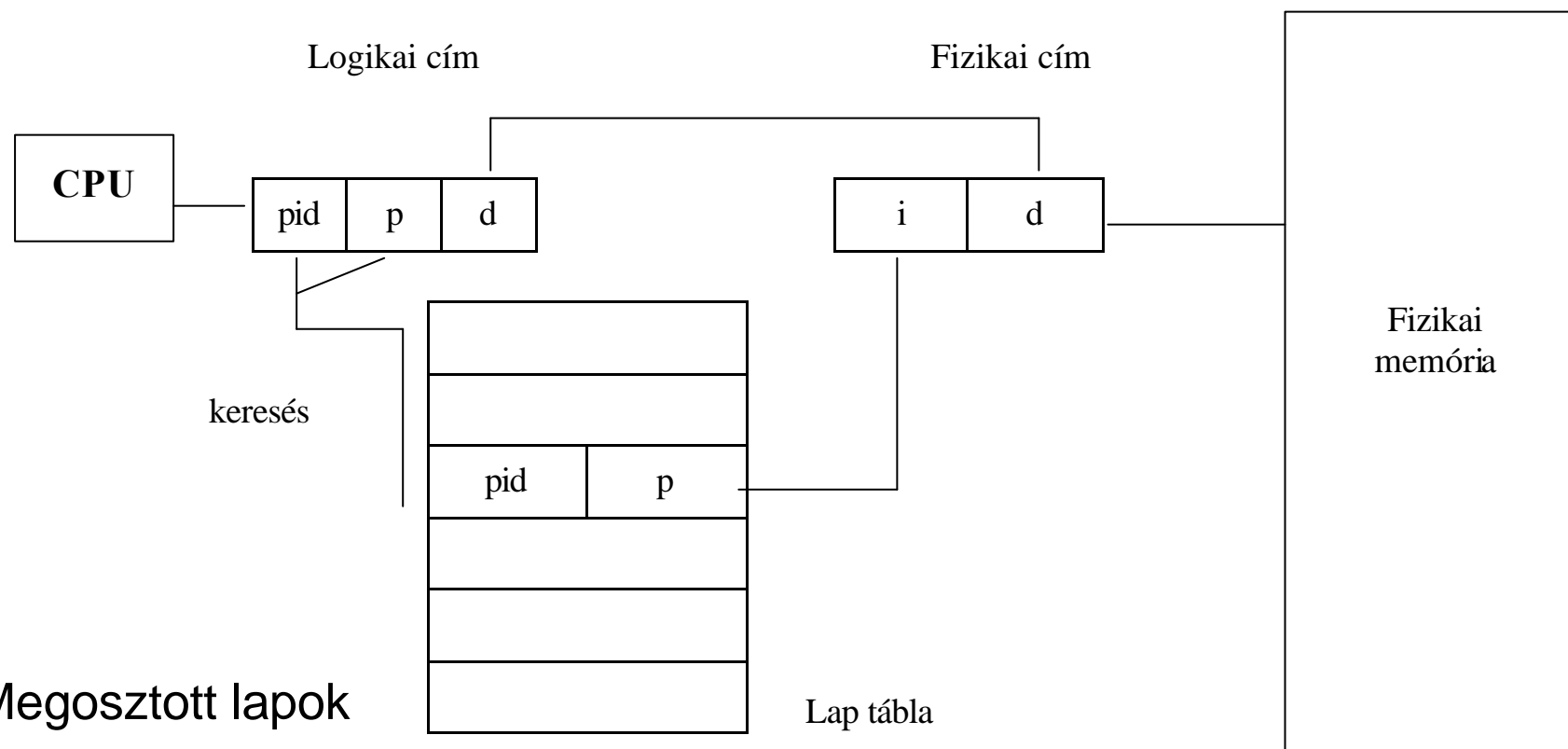
- Az operációs rendszer nyomon követi az allokált és szabad partíciókat (hely és méret alapján).
- Dinamikus tár-hozzárendelési probléma: hogyan lehet kielégíteni egy adott méretű allokációs (tárfoglalási) igényt?
 - **First -fit**: foglaljuk le az első lyukat, amely elég hosszú!
 - **Best-fit**: foglaljuk le az elég hosszú lyukak közül azt, amelynek hossza legközelebb esik a szükséges hosszhoz!
 - **Worst-fit**: foglaljuk le a leghosszabb lyukat (ha az elég hosszú)!
- Értékelési szempontok / értékelés:
 - Sebesség, tárkihasználás: a "first" és "best" jobb, mint a "worst".
 - A lyukak adminisztrálása is időt és memóriát igényel, deadlockhoz is vezethet! (többbe kerülhet mint a szabad hely!)
 - 50% szabály: (D. Knuth) a *first fit* stratégia statisztikai vizsgálata azt eredményezte, hogy n blokk elhelyezése során $n/2$ blokk (helye) elvész(!) (ez a kezdeti szabad terület egy harmada!).
- Külső és belső fragmentáció.
- A külső fragmentáció csökkentése tömörítéssel.
 - cél: a lyukak egyesítése egy szabad területté.
 - dinamikus relokáció szükséges
 - függőben levő I/O problémája.

• Lapozás (paging) – nem folytonos tár-allokálás

- A külső fragmentáció problémájának egy lehetséges megoldását kapjuk, ha megengedjük, hogy a fizikai címtér ne legyen folytonos, megengedve egy processzushoz fizikailag össze nem függő memória blokkok allokálását.
- A logikai- és fizikai címtér független blokkokra (lapokra, keretekre) bomlik.
- A logikai blokk (lap/page) és a fizikai blokk (keret/frame) mérete megegyezik.
 - A méret 2 hatvány, jellemzően 512-8192.
- Bármelyik lap elfoglalhatja bármelyik keretet.
- Nyilván kell tartani a szabad és foglalt kereteket.
- Egy n lapból álló program futtatásához előbb n szabad keretet kell találni!
- Belső fragmentáció.
 - (Külső fragmentáció nincs, mert egy keret mindig teljesen lefoglalódik)
- A címképzés mechanizmusa:
 - **logikai cím:** (lap *sorszáma*, lapon belüli *relatív cím*)
 - **fizikai cím:** (keret *memóriabeli kezdőcíme*, kereten belüli *relatív cím*)
 - **laptábla:** minden logikai laphoz tartalmaz egy bejegyzést, amely a logikai lapot tartalmazó keret fizikai címe + más információk.

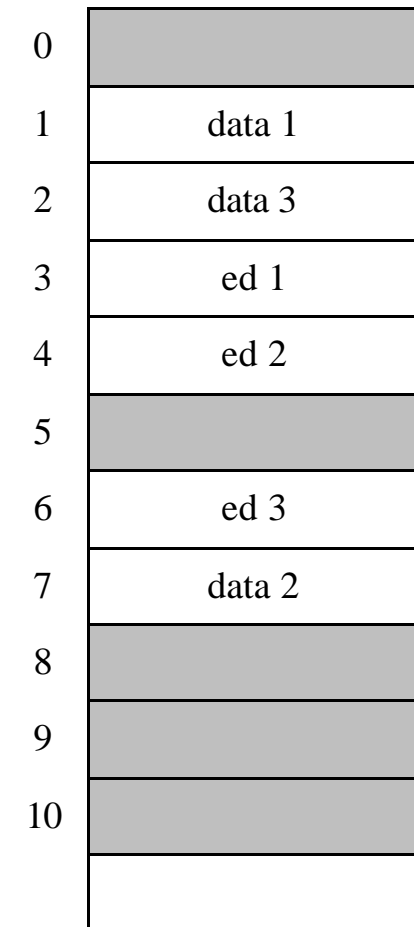
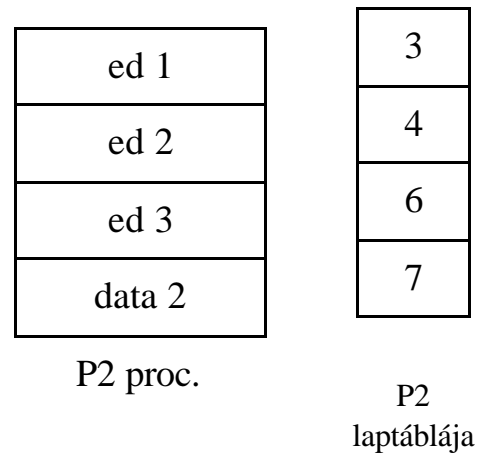
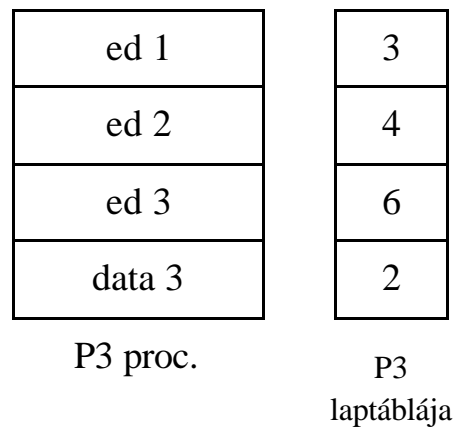
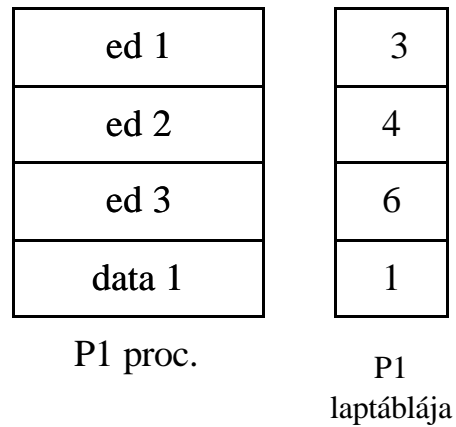


- Minden egyes fizikai laphoz (frame-hez) tartalmaz egy bejegyzést (entry). Egy bejegyzés az adott frame-ben tárolt logikai lap virtuális címét és annak a processzusnak az azonosítóját tartalmazza, amelyikhez a frame tartozik.
- Csökken a laptáblák tárolásához szükséges memória mérete, de nő a tábla átnézéséhez keresési idő a lappreferencia felmerülése esetén.
- Hash - megoldásokkal a keresési idő csökkenthető.



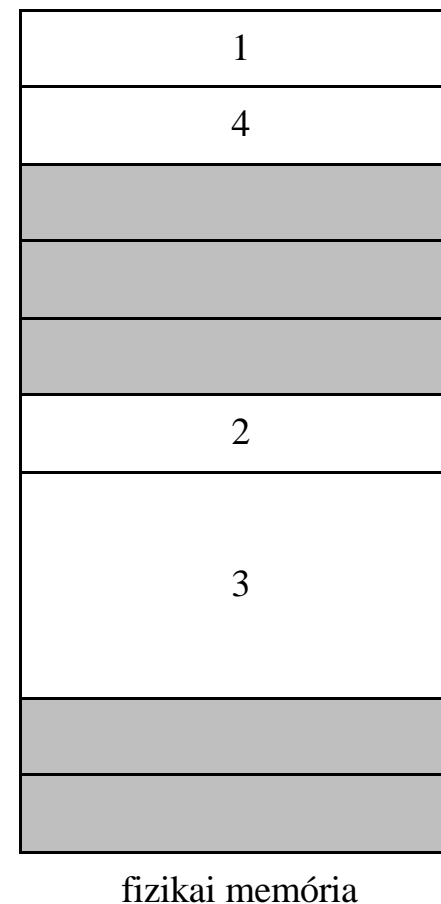
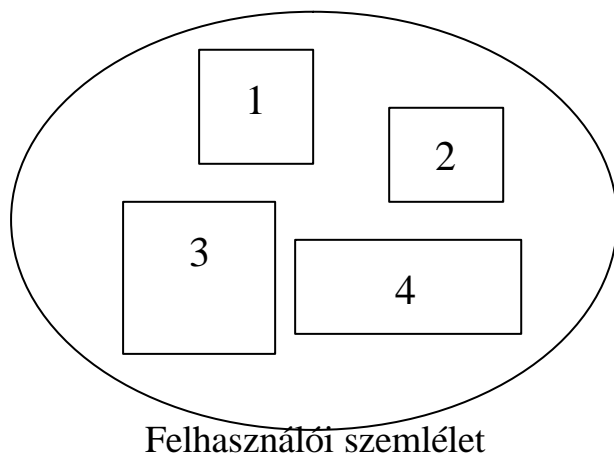
- **Megosztott lapok**

- A közös (re-entrant) csak olvasható kód (lap) megosztva használható több processzus által (pl. szövegszerkesztők, kompájlerok, ablak rendszerek).



• Szegmentáció – szemantikus memória felosztás

- A szegmentáció egy *felhasználói szemléletet tükröző* memória kezelési sémát jelent.
- A program szegmensek együttese. A szegmens logikai egység, mint pl.
 - főprogram
 - eljárás
 - függvény
 - lokális változók
 - globális változók
 - közös változók (common block)
 - verem
 - szimbólum táblázatok, tömbök
- Pl.



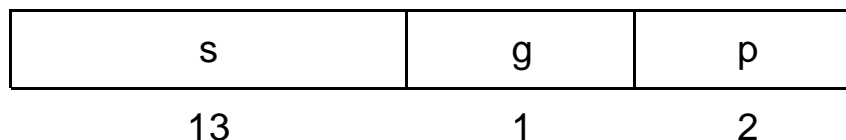
- A logikai cím két részből áll:

< szegmens szám, offset >

- Szegmens tábla – két dimenziós felhasználó által definiált címeket egy dimenziós fizikai címekké alakít; a táblában minden bejegyzés tartalmaz egy
 - bázist – amely a szegmens fizikai kezdőcímét adja meg,
 - mérethatárt (limit), amely a szegmens hosszát mondja meg.
- Szegmens táblázat bázis regiszter (STBR): a szegmens tábla memóriabeli helyére (kezdőcím) mutat (pointer).
- Szegmens táblázat hossz regiszter (STLR): a szegmens tábla maximális bejegyzéseinek számát adja meg.
 - az s szegmens szám akkor legális, ha $s < [\text{STLR}]$.
 - Relokáció: – dinamikus
– szegmens táblázat segítségével
 - Megosztás: – megosztott szegmensek
– azonos szegmens (sor)szám
 - (Tár)védelem: minden egyes bejegyzéshez a szegmens táblában kapcsolódik egy:
 - érvényesítő (validation) bit ($=0 \Rightarrow$ illegális szegmens)
 - read/write/execute privilégiumok
 - Allokáció: – a hosszútávú ütemezőnek el kell helyeznie a memóriában egy processzus összes szegmensét (hasonló megoldások és problémák lépnek fel, mint a változó hosszúságú multipartíciós rendszereknél)
 - first fit /best fit
 - külső fragmentáció

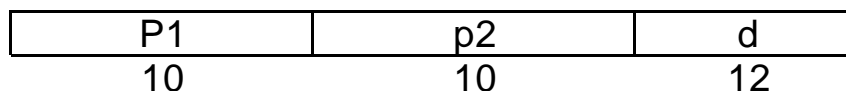
• Szegmentáció lapozással

- **ötlet:** a lapozás a külső fragmentációt, a szegmentálás a belső fragmentációt csökkentheti!
- **PI. INTEL (>3)86** (Az OS/2 operációs rendszer már ki is használta)
 - Egy processzus által használható szegmensek maximális száma: 16K (!)
 - Egy szegmens mérete: 4 GB.
 - Lapméret: 4K=4096 bájt.
 - A szegmensek egyik fele privát, ezek címét (adatait) az LDT (Local Descriptor Table) tartalmazza
 - A többi (az összes processzusok által) közösen használt szegmens, ezek címét a GDT (Global Descriptor Table) tartalmazza.
 - Mindkét táblában egy-egy bejegyzés 8 byte, az adott szegmens leírója (kezdőcím és hossz).
 - Logikai cím: <szelektor, offset>, ahol az offset egy 32 bites érték, a szelektor

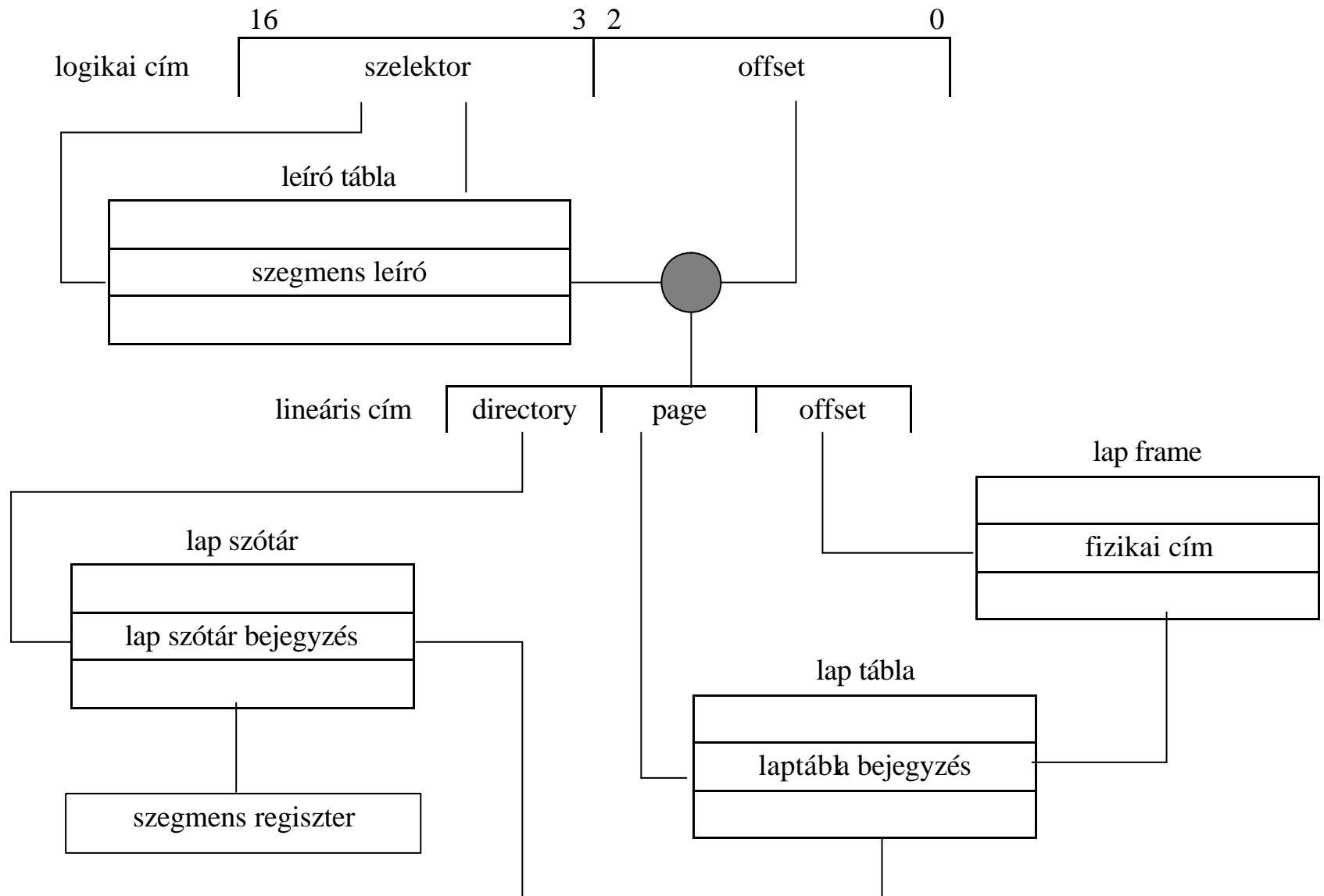


alakú, ahol s: szegmens szám, g: GDT, vagy LDT, p: protection (védelem) jelzése.

- A processzor 6 szegmens regisztere egy-egy szegmens egyidejű gyors megcímezését teszi lehetővé.
- 8 db 8-bájtos mikroprogram regiszter (cache) a megfelelő LDT/GDT bejegyzések tárolására.
- lineáris cím: (ellenőrzések - limit - után) 32 bit, amit fizikai címmé lehet konvertálni.
- lapméret=4K \Rightarrow 1M elemű laptábla (4MB !) \Rightarrow jobb megoldás a 2 szintű laptábla!



• INTEL (>3)86 címképzési séma



• Szempontok a memória management stratégiák összehasonlításához

- A legerősebben meghatározó tényező a hardver támogatás. (A CPU által generált minden egyes logikai címet ellenőrizni kell (legalitás) és le kell képezni valamilyen fizikai címmé. Az ellenőrzés nem implementálható (efficiens) módon a szoftverben.
- A tárgyalt memória management algoritmusok (folytonos allokálás, lapozás, szegmentáció, szegmentáció és lapozás) sok vonatkozásban különböző értékeket mutat. Néhány ilyen vonatkozás:
 - Hardver támogatás: Az egyszerű és multipartíciós sémákhoz elég egy bázis regiszter, vagy egy bázis/limit regiszter pár. A lapozás és szegmentáció leképező táblázatokat is igényel.
 - Teljesítmény: Az algoritmus bonyolódásával a végrehajtási idő is megnő.
 - Fragmentáció: A multiprogramozás szintjének emelésével processzor kihasználtság nőhet, de közben memóriát veszítünk el. (fix partíciók: belső-, változó partíciók: külső fragmentáció).
 - Relokáció: a program (logikai címtartományának) eltolása. Tömörítés és dinamikus áthelyezés.
 - Csere (Swapping): kiegészítő intézkedés arra az esetre, ha a processzus által elfoglalt fizikai memóriát fel kell adni.
 - Megosztás (Sharing): a hatékonyságot növelhetik a több processzus által közösen használt kód és adatok.
 - Védelem (Protection): a processzusok alaphelyzetben csak a hozzájuk rendelt címtartományt használhatják. (A megosztás természetes velejárója.)

9. Virtuális memória kezelés

- Háttér
- Igény szerinti (kényszer) lapozás
- A kényszer lapozás teljesítménye
- Laphelyettesítési algoritmusok
- Frame-k allokálása
- Vergődés (csapkodás, thrashing)
- Kényszer szegmentálás

• Háttér

- Az előző fejezetben tárgyalt memória kezelési technikák csak érintőlegesen tartalmazzák azt az esetet, amikor a processzus logikai címtartománya ténylegesen nagyobb, mint a fizikai címtartomány. (Az overlay és a dinamikus betöltés segíthetik ennek a problémának a megválaszolását, de általában előzetes átgondolást és külön kódolási erőfeszítéseket igényelnek a programozótól.) Ha nem akarjuk a programozót terhelni, akkor korlátozhatjuk a végrehajtható program méretét a fizikai memória méretére, de ez elég szerencsétlen megoldás. Ugyanis:
 - A programok gyakran tartalmazznak olyan (kód)részeket amelyek rendkívüli hibákat/eseteket kezelnek. Statisztikailag tekintve ezek olyan ritkák, hogy ez a kódrész szinte sohasem hajtódik végre.
 - Tömböknek, táblázatoknak, listáknak sokszor olyan sok memóriát allokálnak, amennyire a konkrét esetek nagyrésztében nincs szükség.
 - A program bizonyos ágai, szolgáltatásai rendkívül ritkán aktivizálódnak. (Pl. USA költségvetés egyenlege.)
- Az a lehetőség, hogy egy olyan program végrehajtása is megkezdhető/folytatható, amelynek kódja nincs teljes terjedelmében az operatív memóriában több előnnyel jár:
 - Nem korlát többé a fizikai memória mérete. A programozó nem vesződik az overlay megtervezésével.
 - Egyszerre több program aktív kódrésze lehet benn a memóriában, ami jobb CPU kiszolgálást eredményezhet. (Közben nem növekszik a válaszadási és végrehajtási idő!)
 - Kevesebb I/O tevékenység szükséges a Swapping-hez. (A futási sebesség nő!)
- **A virtuális memória koncepciója a felhasználó/programozó memória szemléletének teljes szeparálását jelenti a fizikai memóriától.**
- **Első közelítésben azt az esetet vizsgáljuk, amikor a futó processzusok fizikai lapoknak (frame-knek) egy rögzített készlete tartozik. (a processzusok nem igényelhetik egy másik processzus fizikai lapjait).**

- Igény szerinti (kényszer) lapozás
- Csak akkor töltsünk be egy lapot a memóriába, ha szükséges
 - Kevesebb I/O szükséges
 - Kevesebb memória szükséges
 - Gyorsabb a válaszidő
 - Több felhasználó jut lehetőséghez
- Szükséges egy lap, ha egy aktuális (logikai) címhivatkozás rá utal.
 - Érvénytelen címhivatkozás \Rightarrow abortálás
 - A lap nincs a memóriában \Rightarrow be kell tölteni a memóriába
- Validációs (*valid/invalid*) bit (v) szerepe a laptáblában: $[v=0] \Rightarrow [\text{page fault}]$
- Page Fault (laphiba)
 - Az első hivatkozás a lapra biztosan laphibát okoz.
 - Az operációs rendszer eldönti, hogy a hivatkozás maga is hibás-e (abortálás), vagy
 - a lap nincs a memóriába betöltve.
 - Üres keret (frame) keresés.
 - Lap betöltése a keretbe.
 - A megfelelő táblaelemek beállítása ($v=1$, stb...)
 - A hivatkozott (gépi) utasítás végrehajtásának folytatása (restart).
- Mi történik, ha egy hiányzó lap betöltéséhez nincs szabad (üres) frame a memóriában?

- Algoritmusok az “áldozat” lap kiválasztására és hatásuk az átbocsátó képességre (cél: a page faultok számának minimalizálása)
- A kényszer-lapozás teljesítő képessége
 - laphiba (page fault) arány: $0 \leq p \leq 1$ ($p=0$: nincs laphiba)
- A “módosított” (*dirty*) bit szerepe a lapcserére fordított idő redukálásában
- Effektív elérési idő (EAT: Effective Access Time)

$EAT = (1-p) * \text{memória elérési idő} +$
 $+ p(\text{ a laphibával kapcsolatos póttevékenység}^*$
 $+ [\text{az áldozat-lap kimentési ideje}]$
 $+ \text{a hivatkozott lap betöltési ideje}$
 $+ \text{újraindítási idő})$

- Példa:

- memória elérési idő (laphiba kizárásával): 1 μsec
- 50%-ban lesz egy áldozat lap módosított, ezért kimentendő
- Lapmentési/betöltési idő: 10 msec = 10000 μsec
- **$EAT = (1-p) * 1 + p * (15000) \gg 1 + 15000 p$** (μsec -ben kifejezve)
- Az EAT-t a lapmentési/ betöltési idő domináns módon meghatározza!

* overhead

- **Laphelyettesítő (áthelyező - replacement) algoritmusok (stratégia, politika)**

- **Fő cél: a laphibák számának minimalizálása.**

- **FIFO (First In First Out) algoritmus**

- Az áldozatot (azaz a memóriát elhagyni kényszerülő lapot) a lapok fizikai sorrendjének megfelelően jelöljük ki!

- Előny: csekély hardver támogatás szükséges!

- **Referencia sztring: 1,2,3,4,1,2,5,1,2,3,4,5**

- 3 frame esete:

1	1	4	5	
2	2	1	3	
3	3	2	4	9 page fault

- 4 frame esete:

1	1	5	5	
2	2	1	4	
3	3	2		
4	4	3		10 page fault(!)

- **Bélády - anomália: több frame \Rightarrow kevesebb laphiba!**

• Optimális algoritmus (OPT/MIN)

- **Létezése:** véges probléma, létezik minimum.
- Azt a lapot helyettesítsük, amelyiket a leghosszabb ideig nélkülözni lehet (mert nem lesz rá szükség!).
- 4 keretes példa:
- **Referencia sztring:** 1,2,3,4,1,2,5,1,2,3,4,5

1	1	4
2	2	
3	3	
4	4	5

6 page faults

- Hogyan lehet implementálni? (Tisztán nem lehet!)
 - Esetleg predikcióval megkísérelhetô az implementáció.
- Hasonlóság az SJF processzus ütemezéshez.
- Elméleti jelentôsége: segítségével az egyes algoritmusok összevethetôk, értékelhetôk.

• Legrégebben használt (LRU Least Recently Used) algoritmus

- Azt a lapot helyettesítsük, amelyiket a leghosszabb ideje nem használunk.
- 4 keretes példa:
- Referencia sztring: 1,2,3,4,1,2,5,1,2,3,4,5

1	1		5	
2	2			
3	3	5	4	
4	4	3		

8 page faults

• Counter (számlálós) implementáció:

- Globális számláló (S): minden memória hivatkozáskor 1-gyel nő.
- Minden f fizikai laphoz tartozik egy S_f lokális számláló, amely minden f -re történő hivatkozáskor felveszi S értékét: $S_f = S$.
- A lokális számlálók értékének összehasonlításával eldönthető, melyik lap legyen az áldozat?

• Verem implementáció:

- A fizikai lapsorszámokat egy verembe helyezzük úgy, hogy a legutoljára használt legyen a tetején (akkor a legrégebben használt lap a verem alján van).
- Előnyei, és hátrányai.

- **AzLRU algoritmust approximáló algoritmusok**

- **Miért lehet jó egy (rossz) approximáló algoritmus?**

- **Referencia bit**

- Minden fizikai laphoz egy bitet rendelünk, amelynek kezdeti értéke: 0.
 - Ha a lapra hivatkozás történik, akkor a bit értéke 1 lesz.
 - Lapcserénél az áldozatok a 0 referenciájú lapok közül kerülnek ki (mindegy milyen sorrendben)
 - Hardver támogatás jelentősége!

- **Kiterjesztett referencia bit: referncia bájt.**

- Minden f fizikai laphoz egy bájtot R_f rendelünk, amelynek kezdeti értéke: $X'00'$.
 - Szabályos időközönként (pl. 100 msec: timer megszakítás!) minden R_f egy bittel (logikailag) jobbra tolódik.
 - Ha a lapra hivatkozás történik, akkor a bájt értéke $R_f = R_f \text{ OR } X'80'$ lesz.
 - Lapcserénél az áldozatok a legrégebbi referenciát tükrözô lapok közül kerülnek ki mindegy milyen sorrendben (véges, diszkrét emlékezet!).

- **Második esély algoritmus**

- A referencia bitet is tartalmazó lapokat ciklikusan vizsgáljuk.
- Ha a referencia bit 0, akkor a lap áldozat lesz.
- Ha a referencia bit 1, akkor lenullázzuk, de a lapot benn hagyjuk (second chance = második esély).
- Helyette következő lapot vizsgáljuk meg az előbbi elveknek megfelelően.
- Speciális eset adódik, ha minden bit = 1. (vö.: FIFO)

- **Kiterjesztett második esély algoritmus**

- A referencia és a módosítás biteket is használja
- Négy lehetőség: (0,0), (0,1), (1,0), (1,1) jelentése. Osztályok
- Az első nem üres osztály első elemét távolítjuk el.
- Alkalmazás: Apple Mcintosh.

- **Számláló (counter) algoritmusok**

- Minden fizikai laphoz tartozik egy számláló (harver implementáció?), amely a lapra történt hivatkozások számát adja meg.

- **LFU (Least Frequently Used) algoritmus**

- Az lesz az áldozat amelyikre a legkevesebb referencia történt.
- Probléma: ha pl. egy inicializáló lapra kezdetben sok hivatkozás történt, mindig benn marad (,mert nagy a referencia száma), pedig többé nincs szükség rá!

- **MFU (Most Frequently Used) algoritmus**

- Az lesz az áldozat amelyikre a legtöbb referencia történt.

- Mindkét esetben az implementáció nagyon költséges!

- **Lap puffereelés**

- pool = frame puffer
- A beolvasás / kiírás frame pufferbe történik (így page fault esetén a processzus továbbindulhat anélkül, hogy várna az áldozat kiírására).
- Más: ha a lapozó eszköz szabad, kiír egy módosított lapot és a dirty bitet nullázza.
- (VAX/VMS – nél alkalmazzák FIFO helyettesítési algoritmussal.)

• Frame-k allokálása

- Alapkérdés: hány frame tartozzon egy-egy processzushoz?
- Minden processzusnak szüksége van minimális számú lapra, hogy futni tudjon. (Ez a szám architektúra függő!)
 - Adott esetben egyetlen gépi utasítás végrehajtásához is több lapra lehet szükség.
 - Pl. IBM370 MVC utasítás (move) végrehajtása 6 lapot is igényelhet!
- Allokációs sémák:
 - Fix allokáció:
 - Egyelő leosztás, minden processzus ugyanannyi frame-t kap.
 - A processzus hosszától függő, arányos leosztás.
 - Prioritásos allokáció:
 - A kapott lapok száma a processzus prioritásától is függ.
 - A kettő keveréke jó megoldás lehet.
 - Egy processzus – page fault esetén – kaphat frame-t egy alacsonyabb prioritású processzus készletéből.
- Globális – lokális allokáció:
 - Globális: új laphoz egy közös készletből lehet.
 - Lokális: minden processzus csak saját, kezdetben hozzárendelt frame készletét használhatja.

• Vergôdés (csapkodás, thrashing)

- Ha egy processzus számára nem áll elég lap rendelkezésre, akkor a **page fault arány** nagyon nagy lesz.
- Következmények:
 - Alacsony CPU kihasználás.
 - Az operációs rendszer úgy gondolja, hogy növelnie kell a multiprogramozás fokát.
 - Új processzust indít.
- Vergôdés – ami ezután jön: a rendszer csak lapok kimentésével és betöltésével van elfoglalva.
- Lokális modell:
 - Lokális: azon lapok halmaza, amelyekre a processzusnak nagyjából egyszerre van szüksége.
 - A processzus vándorol az egyik lokálisból a másikba.
 - A lokálisok egymást átfedhetik.
 - Vergôdés akkor lép fel, ha
$$S \text{ lokálisok mérete} > \text{teljes memória méret}$$
- Working-set (munkahalmaz, munkakészlet) modell

• Kényszer szegmentálás

10. Fájl rendszer interfész

- Fájl koncepció
- Elérési módok
- Könyvtár szerkezet
- Védelem
- Konzisztencia szemantika

• Fájl koncepció

- A számítógépek az adatokat különböző fizikai háttértárakon tárolhatják (pl. mágnes lemez, szalag kártya, optikai lemez, stb.). A számítógép kényelmes használhatósága érdekében az operációs rendszerek egy egységes logikai szemléletet vezetnek be az adattárolásra és adattárakra: az operációs rendszer elvonatkoztatva a tároló eszköz és a tárolt adat fizikai tulajdonságaitól, egy logikai tároló egységet (adatállomány – **fájl** – *file*) használ.
- A fájlokat az operációs rendszer képezi le a konkrét fizikai tároló berendezésre.
- A fájlokat tartalmazó fizikai tároló berendezések általában nem törlődőek (de ez nem kritérium).
 - Pl. "képernyő" fájlok egyes rendszerekben.
- Felhasználói szemszögből: a fájl összetartozó adatok egy kollekciója, amelyeket egy másodlagos tárban tárolunk. A fájl a felhasználó számára az adattárolás legkisebb allokációs egysége: felhasználói adatot a háttértáron csak valamilyen fájlban tárolhatunk.
- Az operációs rendszer támogatást nyújthat a fájl *tartalmának kezelésében*, a fájl *szerkezetének* (adatszerkezet) *létrehozásában*:
 - Fájltypusok: adat – program, folyam (stream) – rekord.

• Fájl attribútumok

- Az operációs rendszer **a fájl kezeléséhez szükséges információkat** (attribútumokat) is tárolja a háttértárban.
 - (Pl. egy elkülönített helyen, a szótárban, fájl jegyzékben – **directory**-ban .)
- **Név** – a fájl azonosítására szolgál (szimbolikus fájlnev, név hossza, osztott nevek, kisbetűk/nagybetűk szerepe)
- **Típus** – az egyes típusokhoz (text, script, bináris program, kép, stb) különböző operációs rendszer szolgáltatásokat lehet rendelni.
- **Lokáció** – a fájl fizikai elhelyezkedésével kapcsolatos adatok (háttértár címek)
- **Méret** – aktuális / maximális méret
- **Védelem** – a fájlhoz történő hozzáférés vezérlése.
- **Idő, dátum, felhasználó azonosítás.**

• Fájl műveletek

- A fájl műveleteket, mint egy **absztrakt adattípuson végezhető műveleteket** kell tekintenünk.
- Az operációs rendszer ezeket **rendszer-hívások** segítségével teszi elérhetővé.
- Alapműveletek:
 1. **Létrehozás** (create) – terület allokálás + új bejegyzés a directoryba.
 2. **Írás** (write) – write pointer szerepe
 3. **Olvasás** (read) – kurrens pozíció szerepe
 4. **Újrapozicionálás** (repositioning)
 5. **Törlés** (deleting)
 6. **Csonkítás** (truncate)
- További műveletek: bővítés (append), átnevezés (rename), másolás (copy), az attribútumok megváltoztatása.
- **Open/close** operációk szükségessége és szerepe:
 - A directory-hoz fordulások számának csökkentése (*open–file table*).
 - Multiuseres környezet támogatása, fájlvédelem (*open–count*).
 - A fájl elhelyezkedésével kapcsolatos információk memóriában tartása (*file pointer, memory mapping*).

• Fájl típusok

fájl típus	kiterjesztés	funkció
végrehajtható (executable)	.exe .com .bin vagy "üres"	futtatható, gépi kódú program
tárgy (object)	.obj .o	lefordított, áthelyezhető bináris kód
forrás (source)	.c .p. .pas .asm	program forráskódja
parancs (batch)	.bat .sh .cmd vagy "üres"	parancs fájl
szöveg (text)	.txt .doc	szöveges adat, dokumentum
szövegszerkesztő (WP)	.doc .rtf .tex .ltn .wp	változatos szövegszerkesztő formátumok
könyvtár (library)	.lib .dll	szubrutin könyvtárak
nyomtatás/megjelenítés (print/view)	.ps .pdf .gif .prn	megjelenítő berendezések saját fájlformátumai
archívum (archieve)	.arc .zip .tar	kapcsolt fájlok egy fájlba szervezve, esetleg tömörítve
alkalmazói rendszerek (appl. syst.)	???	

• Fájl szerkezetek

- A fájl típus indikál(hat)ja a fájl (belső) szerkezetét. (A kreátornak és a fájlt feldolgozó programnak ezt természetesen ismernie kell.)
- Pl. forrásnyelvi-, tárgynyelvi- (object), ill. végrehajtható (bináris) programokat tartalmazó állományok szerkezete kötött.
- Kérdés: Mennyire ismerje az operációs rendszer az egyes fájlok belső szerkezetét?
- Egy szerkezetet (végrehajtható program) biztosan ismernie kell!
- Egyébként olyan mértékben ismeri, amennyire a fájlkezeléshez "központi támogatást" akar nyújtani.
- Pl.:
 - Néhány régi nagy rendszer (IBM DOS 26.2; IBM OS/VS, stb.) támogatta a változatos blokkolási és fájl szervezési módokat (FIXUNB, FIXBLK, VARUNB, VARBLK, UNDEF; VSAM, VDAM, ISAM).
 - A UNIX minden fájlt bájtok egyszerű sorozatának tekint, ez a feldolgozó programoknak nagy flexibilitást, de minimális támogatást jelent. Szolgáltató ugyanakkor egy mechanizmust (rendszer hívás) a fájl típus meghatározására. (Vö.: *file* parancs, *magic number*)
 - VAX/VMS három szerkezetet is támogat!
- Az operációs rendszer által nyújtott szerény támogatást kiegészíthetik a programozási környezetekben / futtató rendszerekben implementált szerkezet-kezelési lehetőségek.
 - (Pl. adatbázis kezelők.)

• Elérési módok

• Szekvenciális elérési mód

- read next
- write next
- reset
- no *read* after last *write*
- (rewrite)

• Direkt elérési mód

- read n
- write n
- position to n (pozicionálás az n-edik rekordra)
- read next
- write next
- rewrite n

n= a relatív blokk-sorszám

• Könyvtár szerkezet

- Bejegyzések (node) együttese, amely információt tárol a fájlokról
- A könyvtárszerkezet és a fájlok is a lemezen a háttértáron tárolódnak.
 - Kapcsolódó fogalmak: kötet (volume), partíció (partition), VTOC
- Egy directory-ban (fájl-jegyzékben) tárolt információ:
- (pl.) név, típus, cím, aktuális hossz, maximális hossz, legutóbbi elérés (access) időpontja, legutóbbi módosítás (update) időpontja, tulajdonos azonosító (owner ID), védelemmel kapcsolatos adatok.
- Egy directory-val kapcsolatban végrehajtható (absztrakt alap-) műveletek:
- Fájl keresés (file search), fájl létrehozás (create), fájl törlés (delete), directory listázás (dir, list, ls), fájl átnevezés (rename), a fájl rendszer pásztázása (traverse), bejárása.
- A könyvtárszervezéssel szemben támasztott elvárások:
- Hatékonyság: minden fájl könnyen visszakereshető legyen.
- Névadás (naming):
 - Két user használhassa ugyanazt a nevet más fájlokhoz.
 - Ugyanannak a fájlnek lehessenek különböző azonosítói.
- Csoportosítás (grouping)
 - Fájlok csoportosítása tulajdonságaik alapján. (com, bat, pss, nfs ntfs, ... dtv)

- **Egyszintű directory**
 - – Az összes felhasználó állományai egyetlen *jegyzéket* (directory) alkotnak.
 - – Névadási- (névütközési-) és csoportosítási problémák
- **Kétszintű directory**
 - – Egy-egy felhasználó állományai elkülönített *jegyzéket* (directory) alkotnak.
 - – Szintek: MFD - UFD (master/user file directory)
 - – Megoldja a névütközés problémáját, de nincs csoportosítás.
 - – A felhasználók nem tudnak kooperálni (nehéz egymás állományait elérni).
 - – Új fogalom jelenik meg: *elérési út* (path)
 - – A rendszer fájlok használatának problémája
 - dedikált kópia?
 - egy (több) speciális (kitüntetett) UFD-ben lehet (pl.) elhelyezni. (Primitív csoportosítás.)
 - *Keresési út* (search path) fogalma.
- **Fa-szerkezetű directory**
 - Egy fájljegyzék bizonyos elemei lehetnek újabb fájljegyzékek, így fájljegyzékeknek egy hierarchikus rendszere jön létre.
 - Egy fájljegyzék által (fájlként) tartalmazott fájljegyzék = **alfájljegyzék** (subdirectory)
 - A jegyzékben minden esetben egy speciális bit jelezheti, hogy fájlról, vagy aljegyzékről van-e szó.
 - Fájljegyzéket létrehozni és törölni speciális rendszer-hívásokkal lehet.
 - Pl. *make*: mkdir(); *remove*: rmdir().
 - Kurrens (current) directory, kurrens directory váltás: cd().
 - Abszolút- és relatív elérési út, keresési utak, pásztázás (*traverse*).
 - Megoldja a névütközés problémáját, és lehet csoportosítani.
 - Az elérési utak megadása sokszor körülményes. Egy megoldás: *DeskTop file* (Mac).

- Aciklikus gráf - szerkezetű directory

- Egy aljegyzék (fájl) osztott használatának problémája: több felhasználó/alkalmazás szeretné a saját directory rendszerében látni.
- Megoldás: egy típusú directory bejegyzésnek, valamely fájlra, vagy aljegyzékre mutató kapcsolónak (*symbolic link*) a bevezetése. Logikailag: alias-képzés!
 - Vö: link, ln rendszerhívás (UNIX), create link (NT).
 - Technikailag lehetséges lenne a mutató (link) helyett a teljes directory bejegyzést **duplikálni**:
 - (DE: konzisztencia!)
- *Hard* link és *soft* link.
- Ha a kapcsolókat követve nem juthatunk vissza egy olyan bejegyzéshez, amelyet már korábban elértünk, akkor a directory szerkezete aciklikus gráf.
- Nehéz detektálni a ciklust.
- Problémák: pásztázás, törlés.
- Törlési politikák: Mi történjék a fájlra mutató linkekkel, ha töröljük a fájlt?
 - ..., illetve mi legyen a fájlal, ha töröljük a linket?
- "Függő" link megengedett: megmarad a link, ha töröljük a fájlt (vagy a fájl nem érhető el).
 - Pl. UNIX soft link, lokális lemezek mountolása egy fájl rendszerbe. (Ez jól is jöhet!)
- "Függő" link nem megengedett.
 - Megoldások:
 - fájl-referencia listák
 - referencia számlálók

- Általános gráf - szerkezetű directory
 - Hogyan lehet a ciklusmentességet biztosítani?
 - Vannak algoritmusok, de ezek "költségesek".
 - Ha a szerkezet ciklust (önreferencia) tartalmaz, akkor a keresés / pásztázás körülményes lesz.
 - Pl. egy fájl a körbe mutató linkek miatt nem lesz simán törölhető. Adott esetben "hulladékgyűjtés" (garbage collection) is szükséges lehet.

• Védelem

- A számítógépes rendszerben tárolt adatokat védeni kell
 - a fizikai sérülésektől (*reliability, integrity*), és
 - az illetéktelen hozzáféréstől (*protection, security*).
- A fizikai sérülések ellen a gondos kezelés mellett meghatározott stratégia szerint készülő biztonsági másolatokkal védekeznek.
- Hozzáférési operáció (alap-) típusok:
 - **Read** (olvasás a fájlból)
 - **Write** (írás a fájlba)
 - **Execute** (a fájl betöltése a memóriába és végrehajtása, futtatása)
 - **Append** (új adat hozzáírása a fájl végéhez)
 - **Delete** (a fájl törlése és az általa elfoglalt hely felszabadítása)
 - **List** (a fájl nevének és attribútumainak listázása)
 - Más operációkat (**rename, copy, edit**) is lehetne még tekinteni, ezek visszavezethetők alap- operációkra.

A hozzáférési operációk háttérében az operációs rendszer, vagy egyes felhasználók által kezdeményezett processzusok állnak. Ha tehát egy-egy adott fájlhoz minden egyes processzusra (userre) megmondjuk, hogy melyik alapoperációt alkalmazhatja a fájlra, akkor a hozzáférés teljesen szabályozott lesz. Ez a

- Hozzáférési lista (Access Control List, ACL)

Az ACL megnöveli a fájlbejegyzés méretét, változó hosszúságú lesz, nehezen kezelhető (de alkalmazzák!).

- **Megoldás csoportosítással**

- a védelem szempontjából az összes felhasználó három kategóriába tartozhat:
 - ő maga a fájl tulajdonosa (**owner**),
 - tagja valamilyen jól definiált csoportnak (**group**, több csoport is lehet),
 - beletartozik az összes felhasználók csoportjába (**world**).
- a hozzáférési alapoperációk közül is hármat emelünk ki (a többit ezek mögé illeszthetjük)
 - read (r), write (w), execute (x)
- A hozzáférési lista ekkor leegyszerűsödik: egy-egy fájlra azt kell megmondani, hogy az egyes kategóriák mely operációkat hajthatnak végre a fájlon.
- UNIX példa: (az egyes oszlopok jelentése a `man ls` paranccsal lekérdezhető!)

```
fazekas@paris [~] >>ls -l
```

```
total 354
```

```
drwxr-xr-x  8 fazekas  pcpc    512  Sep  7 14:50  ./
drwxr-xr-x 159 root    root    4096  Nov 23 14:03  ../
-rw-----  1  fazekas  pcpc    496  Sep  7 14:50  .Xauthority
-rwxr-xr-x  1  fazekas  pcpc   2463  Nov 20 1998  .alias*
drwx----- 37  fazekas  pcpc   1024 May  3 1998 .fm/
-rw-----  1  root     other     0    Sep 17 1996 .homedir_angelegt_am_96.09.17_12:24:47
drwx-----  3  fazekas  pcpc    512  Jul 21 1997  .netscape/
drwxr-xr-x  2  fazekas  pcpc    512 Sep 17 1996  .wastebasket/
```

- Más védelmi megoldások:
 - **jelszó** (password) **a fájlhoz** (ki tudja megjegyezni?)
 - archiváló (tömörítő), workflow (office) rendszerekben alkalmazzák
 - **jelszó a könyvtárakhoz** (mindent, vagy semmit!)
 - Pl.: TOPS-20, IBM VM/CMS (minidisk védelem!)
 - **speciális attribútumok**: "read only", "system" (MS DOS)
- A fájljegyzékek, (directory) védelme
 - Külön probléma:
 - Pl. attól még hogy egy, a jegyzékben levő fájl olvasható, nem biztos, hogy a jegyzék tartalmát lehet listázni, vagy
 - nem biztos hogy egy könyvtár kurrens könyvtárrá tehető.
- Unix példa:
- directory esetén (ls által adott listán az első betű d)
 - r jelentése: a directory tartalma listázható
 - x jelentése: cd (change dir) az adott jegyzékre vonatkoztatva megengedett

• Konzisztencia szemantika

- Multiprogramozásnál előfordul(hat), hogy több processzus közösen használ egy-egy állományt. (konkurens hozzáférés). Eközben biztosítani kell az állományban tárolt adatok konzisztenciáját (elkerülni az összeférhetetlenséget). Ehhez egy pontos szemantika (értelmezés) szükséges, amely világossá teszi, hogy az adat milyen állapotot tükröz.
- Röviden: hogyan és mikor válik láthatóvá, elérhetővé egy processzus számára az állományon másik processzus által okozott változás?
- Fogalom: *fájl szeansz/ülés (file session)* = a processzus életének az a szakasza amely a fájl megnyitásától a fájl lezárásáig terjed. (vö. *terminal session*: logintól logout-ig terjedő időszak!)
- UNIX fájl rendszer szemantika:
 - Egy megnyitott fájlba történő írás azonnal látszik a többi felhasználóknak is, akik a fájlt nyitva tartják.
 - Bizonyos feldolgozó programok ezt lokálisan módosíthatják (lock, dedikált kópia használata.)
 - Még olyan megosztás is van, ahol a kurrencia pointer is közös és mindkét fél által mozgatható.
- Szekció szemantika (Andrew - fájl rendszer)
 - Egy megnyitott fájlba történő írás nem látszik a többi felhasználóknak, akik a fájlt nyitva tartják.
 - Csak azok látják majd akik azután nyitották meg, amikor az író processzus lezárta.
 - E szemantika szerint a fájlnek több másolata lehet egy időben.
- Osztott megváltoztathatatlan (*immutable*) fájl szemantika
 - Egy ilyen fájl neve védett, tartalmát nem lehet módosítani. Osztott rendszerben egyszerűen implementálható (read only)

11. Fájl rendszer implementáció

- Fájl rendszer struktúra
- Allokációs módszerek
- Szabad hely kezelés
- Directory implementáció
- Hatékonyság és teljesítmény
- Helyreállítás

• Fájl rendszer struktúra

- A fájl rendszer rétegekbe (layer) szervezhető, ezek
 - *Alkalmazói program*
 - A fájl létrehozásával, olvasásával, módosításával kapcsolatos igények forrása.
 - *Logikai fájl rendszer*
 - A fájljegyzék (directory) felhasználásával, a fájl nevéből kiindulva ez határozza meg a szükséges információkat a szervezési modul számára. Felel a fájl védelméért.
 - *Fájl szervezési modul*
 - Ez ismeri a és implementálja a kapcsolatot a logikai és fizikai rekordok között. Ismerve a fájl szervezési módját és allokációját a logikai rekord címeket fizikai blokkokra történő hivatkozássá alakítja. Adminisztrálja a szabad helyeket.
 - *Alap fájl rendszer*
 - Ez alakítja ki és adja ki a megfelelő device drivernek a megfelelő magas szintű parancsot egy blokk írására és olvasására. (cilinder, sáv, szektor logikában "gondolkodik".)
 - *I/O vezérlés*
 - Device driverek, interrupt handlerok. ezek inputja magas szintű utasításokból (pl. "olvasd a 12. blokkot") áll, outputját alacsony szintű, hardver specifikus bitképletekből alkotják (, amelyeket különböző "portokra" ír).
 - *Fizikai berendezés*
- Fájl megnyitás és lezárás:
 - rendszer és processzus szintű Open File Table, Fájl vezérlő blokk (FCB)
 - UNIX: aktív INODE tábla
- Fájl rendszer mountolás

• Allokációs módszerek

folytonos

indexelt

FAT

UNIX I-NODE

• Szabadhely kezelés

bit térkép

láncolás

csoportosítás (grouping)

számlálás (counting)

Directory implementáció

szekvenciális szerkezettel
hashing

- Hatékonyság és teljesítmény

ható tényezők: paraméterek, allokáció, stb.

• Helyreállítás

Biztonsági másolatok készítése, stratégiák

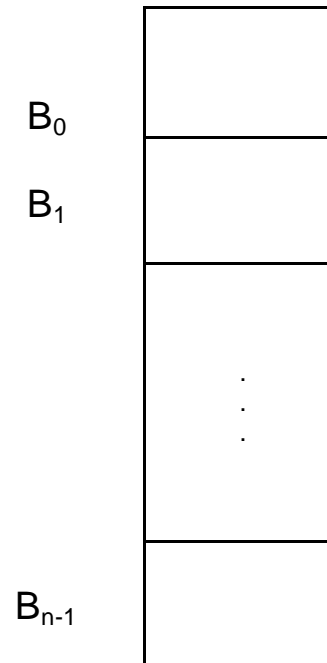
Markov megállítási probléma

12. Másodlagos tár szerkezet

- Diszk felépítés
- Diszk ütemezés
- Diszk kezelés
- Swap (csere) terület kezelés
- Diszk megbízhatóság
- Stabil-tár implementáció

• Diszk felépítés

- Logikailag a diszk blokkokból képezett lineáris tömbnek fogható fel.



- Létezik egy leképező séma, amely a B_i logikai címhez fizikai lemezcímet (cilinder, sáv, szektor) rendel.
- Legkisebb allokációs egység a blokk.
- A blokk egyben a belső fragmentáció viszonyító mértéke.

• Diszk ütemezés

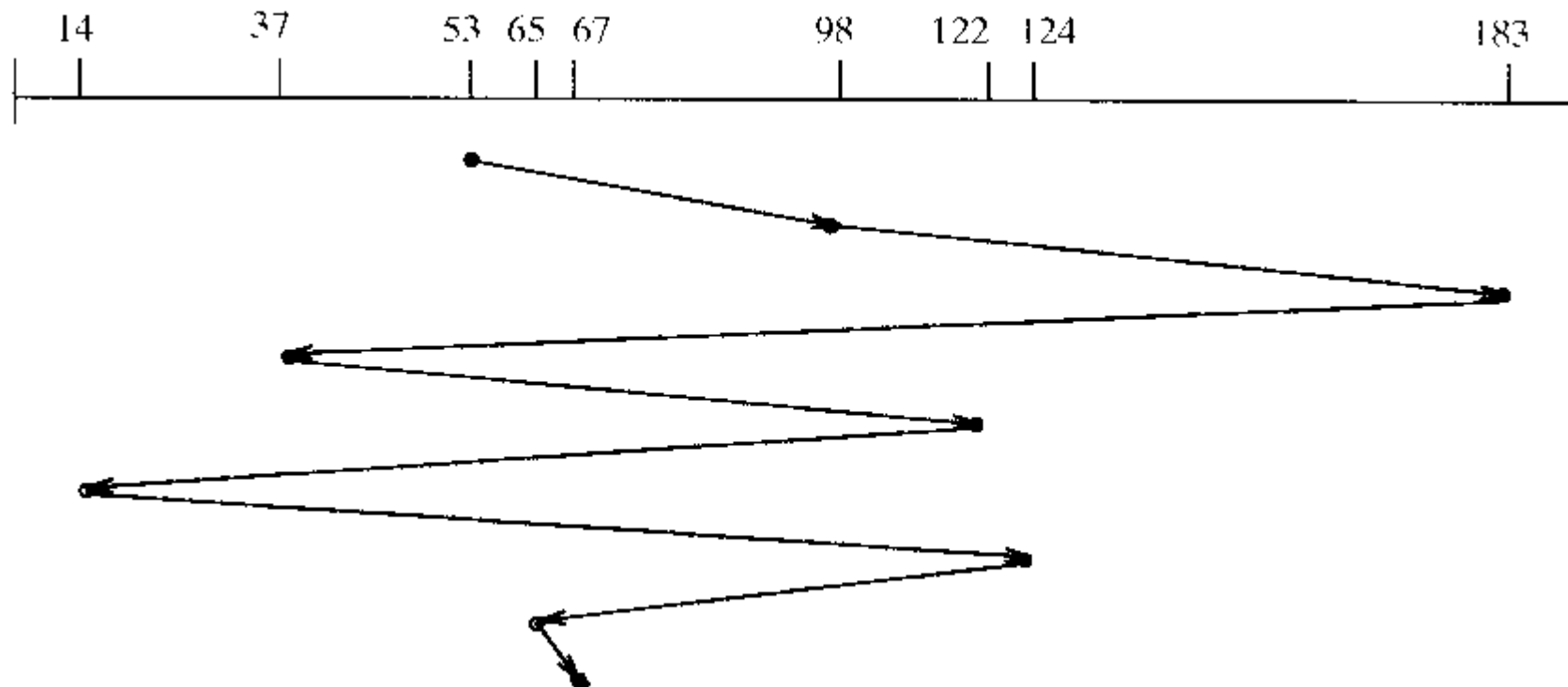
- A lemezművelet mindig valamelyik **sáv** valamelyik **szektorára** történő **pozicionálással** kezdődik:
 - keresési idő (**seek time**): az író/olvasó fej pozicionálása a sávra
 - várakozási idő (**latency time**): várakozás, amíg a kérdéses szektor az író/olvasó fej elé fordul
 - átviteli idő (**transfer time**): az adat írása / kiolvasása elektronikusan
- A keresési idő nagyságrendekkel lehet nagyobb, mint a másik kettő, ezért a diszk ütemezés célja a keresési idő minimalizálása.
 - A keresési idő arányos a keresési távolsággal (jó közelítéssel).
- Ütemezési algoritmusok (példákon illusztrálva).
 - Tegyük fel, hogy egy lemezegység 200 cilindert (0–199) tartalmaz, az író/olvasó fej az 53-ik cylinderen áll és hogy az I/O sorban levő **igények** rendre a

98, 183, 37, 122, 14, 124, 65, 67

sorszámú cylinderekkel kapcsolatosak.

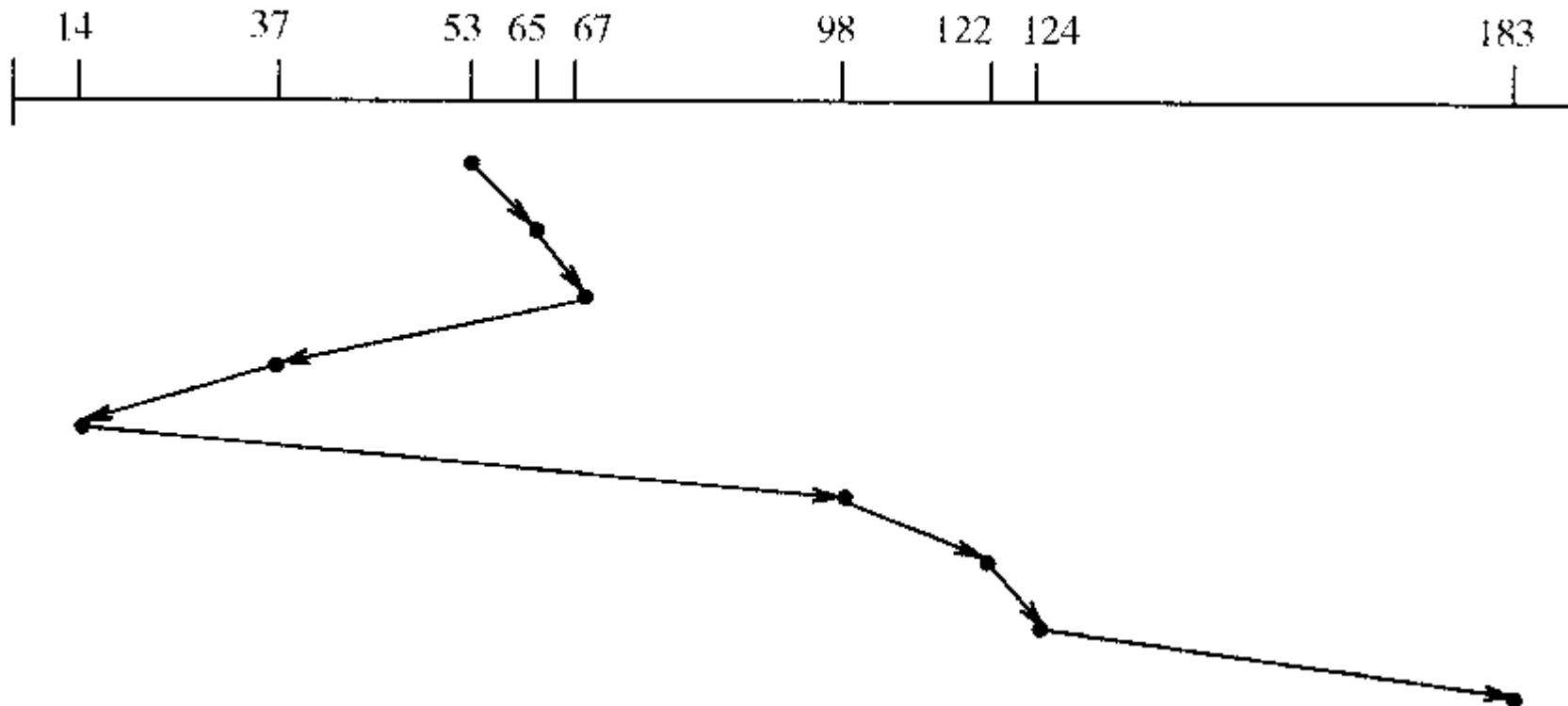
- FCFS (First Come First Served)

- Egyszerûen implementálható, de nyilván nem szolgáltat jó átlagos kiszolgálási időt.
- Az író/olvasó fej ide-oda "ugrál".
- A fej által összességében megtett "út" hosszú.
- Minden igény előbb-utóbb kielégül.

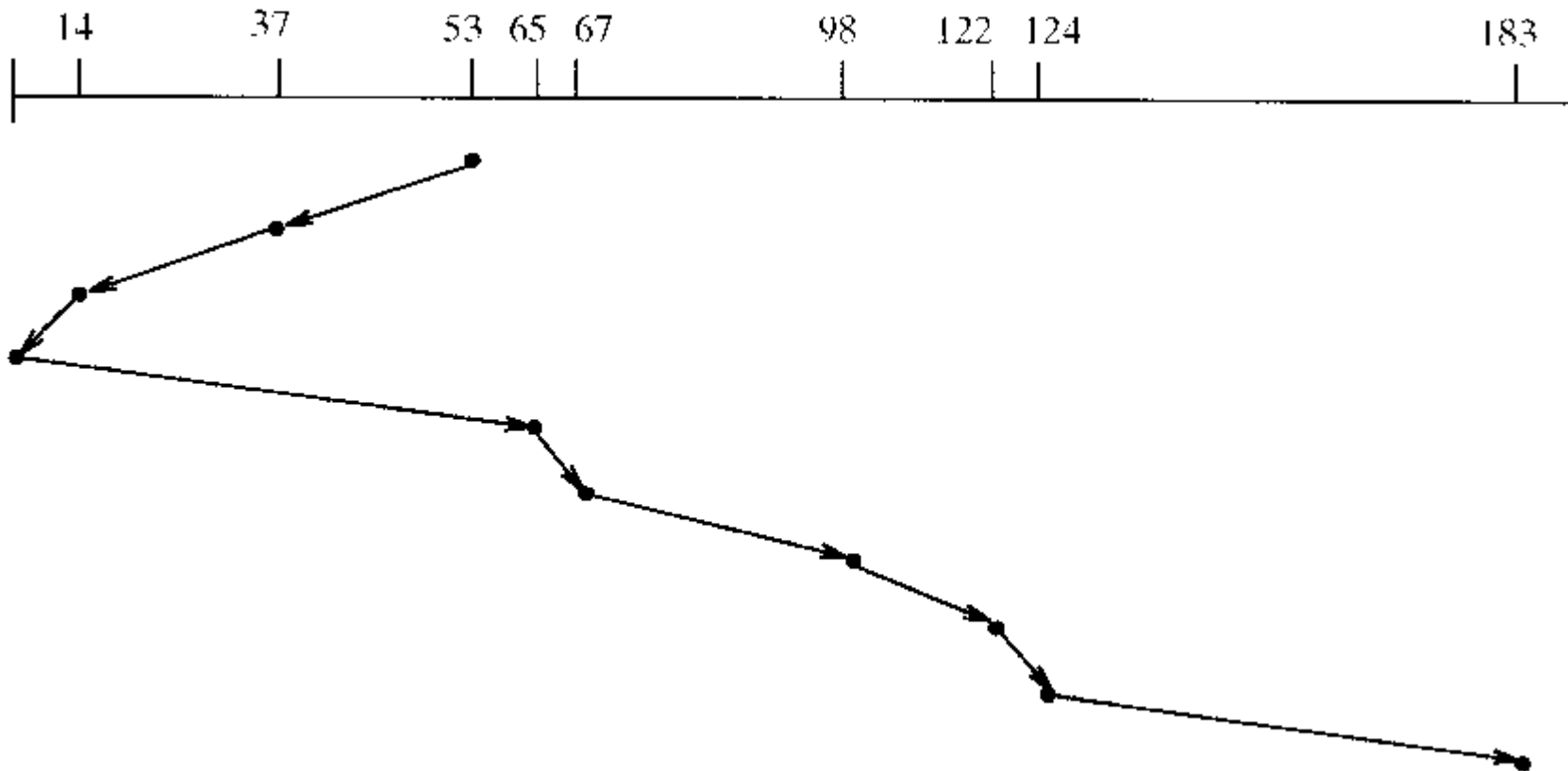


- SSTF (Shortest Seek Time First)

- Hasonlít az SJF processzus ütemezési stratégiára (az ott optimális volt, ellenben ez itt)
- Nem optimális: az **53, 37, 14, 65, 67, 98, 122, 124, 183** sorrend rövidebb összkiszolgálási időt eredményez.
- Az FCFS-nél azért kedvezőbb.
- Éhezéshez (starvation) vezethet!



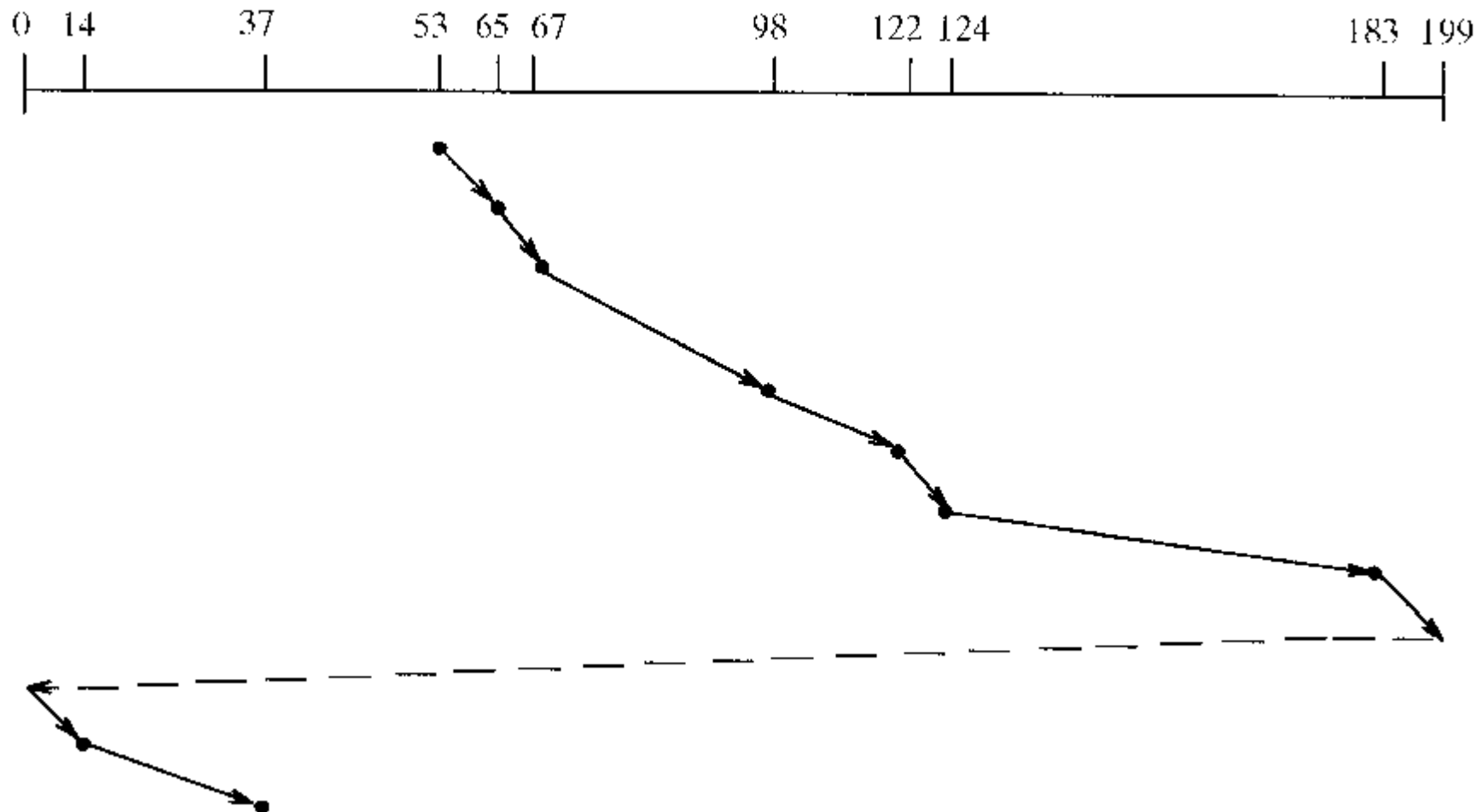
- SCAN (pásztázás)
 - Elevátor algoritmusnak is nevezik.
 - Ha a fejpozicionálási igények egyenletesen oszlanak el, akkor mindig amikor a fejmozgás irány a legszélső cilindernél megfordul, a másik végén van a "tumultus". ? egyenetlen várakozási



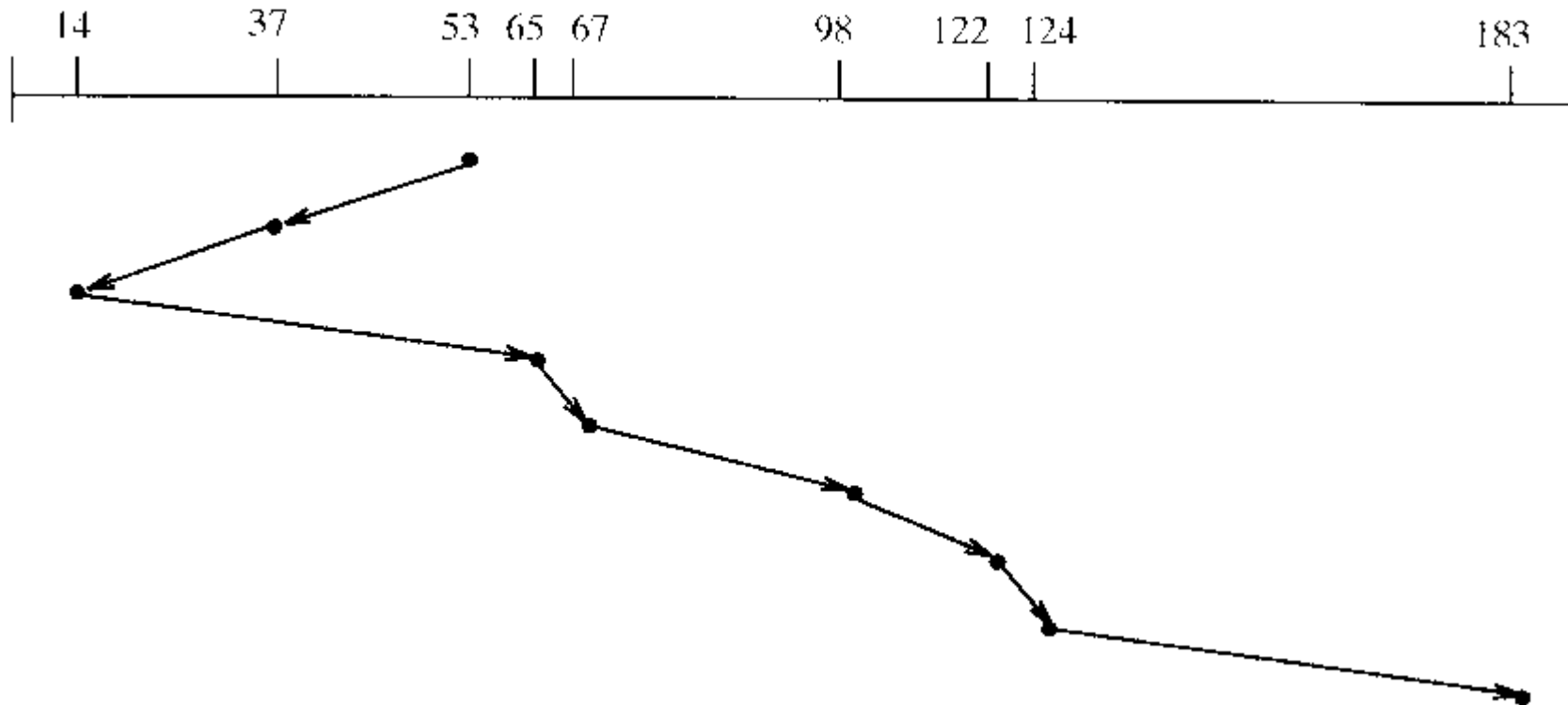
idő

- C-SCAN (Circular SCAN)

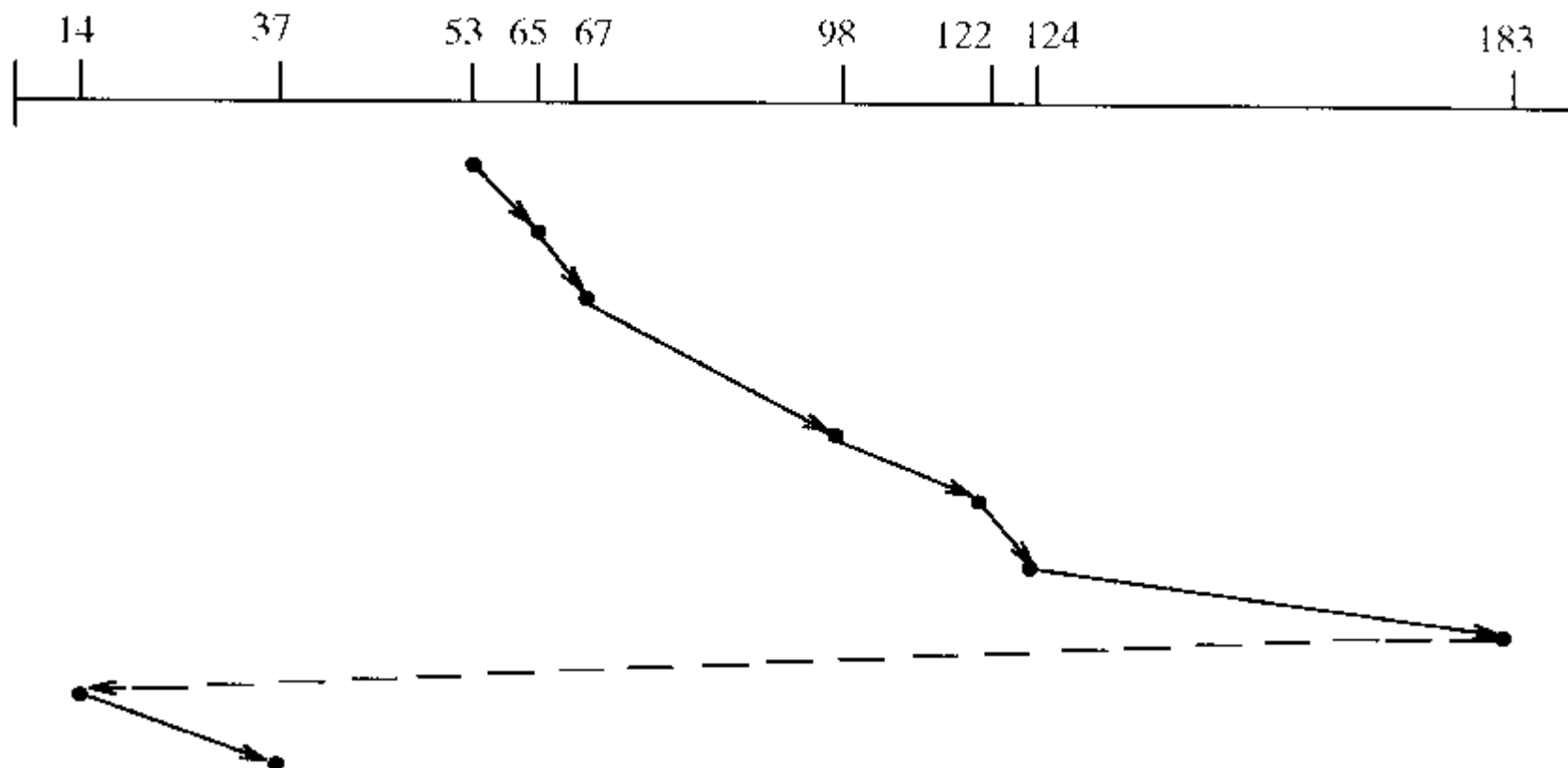
- Az egyenetlen várakozási idő egyenletesebbé tehető vele.



- **LOOK** (for a request before moving)
 - Csak akkor megy valamelyik irányba tovább, ha tényleg van rá igény.



- C-LOOK
 - A LOOK cirkuláris változata



- A lemez - ütemező algoritmus megválasztása
- Az SSTF algoritmus egyszerű és "csábító".
- A SCAN és C-SCAN alkalmasabbak olyan rendszerekben, ahol a lemezes adatforgalom nagyon intenzív.
- Lehetne definiálni az optimális algoritmust, de nem biztos, hogy a nagy számítási munka megtérül.
- Minden alkalmazott algoritmus végső teljesítményét alapvetően befolyásolja az *igények típusa és száma*.
- Pl. ha a sor általában egyelemű, akkor minden algoritmus ekvivalens, így elég az FCFS.
- Fontos befolyásoló tényező a fájlok allokációs módja.

lemez terület kihasználása \Leftrightarrow hozzáférési idő

- Fájljegyzékek (directory), továbbá indexek helye .
- Következmény: szükség van az operációs rendszerben egy elkülönített *diszk ütemező modulra*.

Napjainkban ezt a tevékenységet egyre inkább a lemez vezérlő egységébe integrálják, így az operációs rendszerben elég ha van egy FCFS ütemező.

• Diszk kezelés

A diszk kezelésnek több olyan aspektusa van, ami az operációs rendszer hatáskörébe tartozik.

- **Diszk formázás.** (NEM "formattálás"!!!)
 - Fizikai formázás
 - sávok (track), szektorok helyének kialakítása a homogén (mágnesezhető) lemezfelületen.
 - header (sávcímmel) + adat helye + ECC (Error Correcting Code) helye
 - Ma: gyári fizikai formázás
 - Logikai formázás
 - Lemezcímke (Volume Label), kezdeti üres fájljegyzék, FAT, boot rekord, stb felírása az operációs rendszer által meghatározott formában.
 - Adatbázis kezelők sokszor maguk alakítják ki a logikai szerkezetet.
- **Boot rekord, boot blokk. Bootstrap rendszer**
- **Hibás blokkok kezelése.**
 - IBM-PC MS DOS
 - IDE=Integrated Drive Electronics (manuálisan) A **format** parancs megjegyzi a FAT-ban ha hibás szektort talált. Később csak egy manuálisan elindított program (**scandisk**) tud újakat hozzávenni.
 - SCSI= A kontroller egy listában nyilvántartja a hibás blokkokat és ezt naprakészen tartja. A hibás blokkok helyett saját (az op rendszer által el nem érhető) készletéből jó blokkot ajánl fel. (slipping, szektor átirányítás, ha lehet cilinderen belül.) Baj lehet az ütemezővel!

• Swap (csere) terület kezelés

- A swap terület használata
 - teljes processzus területének kitárolása
 - egyes lapok kitárolása
- A swap terület mérete
- A swap terület allokálása
 - Fájl rendszeren belül (Windows)
 - Külön partícióban (UNIX), és nem a fájl rendszer elérési mechanizmusait használva.
 - Külön partícióban a fájl rendszerből kiegészítve (Solaris)
- Swap terület kezelés, kezelési stratégiák
 - kód és adatszegmensek különválasztása
 - swap map-ek

• Diszk megbízhatóság

- A diszk a számítógépes rendszer legsérülékenyebb részének tekinthető.
- Disk striping: a blokk egyes részei külön lemezekre (szinkronizáció?)
- RAID (Redundant Array of Inexpensive Disks).
 - Paritás diszkek

• Stabil-tár implementáció

- stabil tár = soha nem vesz el róla az adat
 - pl. a *szinkronizációnál* a *write ahead log* kezelésére ilyen kellene
- A tökéletes háttértár szimulációja két lemezzel.

13. Védelem

- A védelem célja
- Védelmi tartományok
- Hozzáférési mátrixok (access matrix, AM)
- A hozzáférési mátrixok implementációja
- A hozzáférési jogok visszavonása
- Képesség-alapú rendszerek
- Nyelvbe ágyazott védelem

• A védelem célja

- A számítógépes rendszer objektumok (hardver és szoftver) együttesének tekinthető.
- Minden objektumnak egyedi neve van és operációknak (műveleteknek) egy jól definiált halmazával érhető el.
 - Absztrakt adattípusok szerepe a modellezésnél!
- Védelmi probléma – biztosítani, hogy minden objektum korrekt módon legyen elérhető, és kizárólag azon processzusok által, amelyeknek az elérés megengedett.
 - need-to-know elv
 - mechanizmusok és politikák szétválasztása

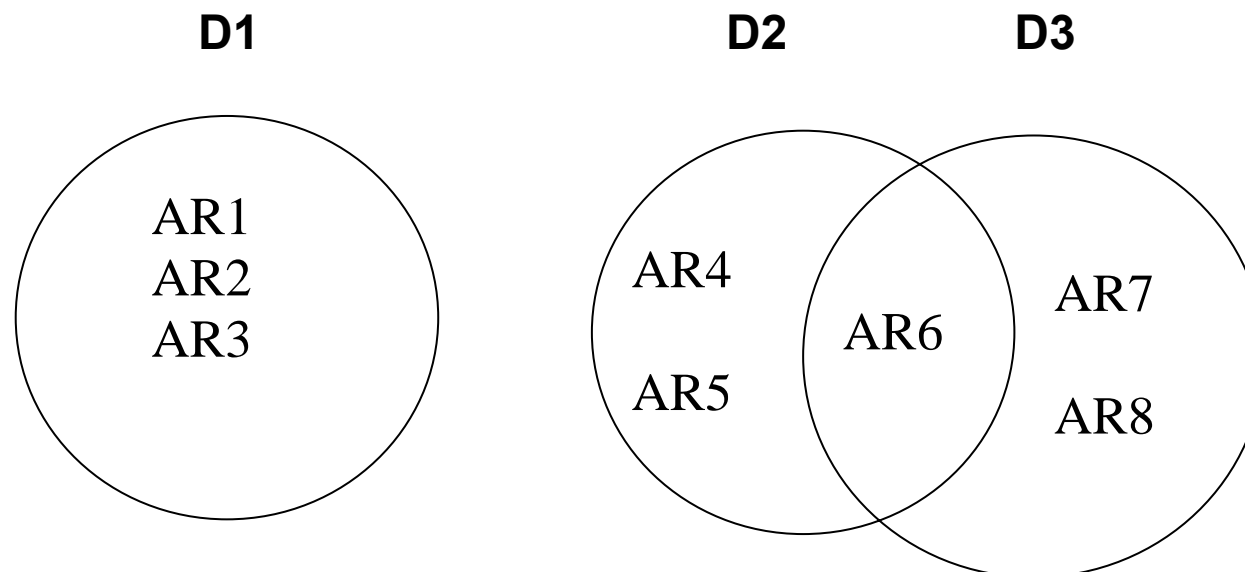
- **Védelmi tartományok** (protection domain)

- **Tartomány szerkezet**

- **Hozzáférési jog** = **<objektum név, joghalmaz>**

A joghalmaz a rendszerben implementált (érvényes) operációknak az a részhalmaza, amelynek elemei az objektumra végrehajthatók.

- **Tartomány** = **hozzáférési jogok halmaza**



AR1 = <file_A, {read, write}>

AR8 = <file_A, {read}>

- Tartomány implementáció

- **A rendszerben két *tartomány* van** (primitív)

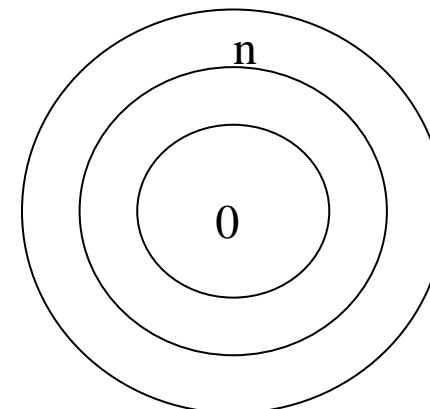
- user
- supervisor

- **UNIX**

- *Tartomány* = USER-ID
- *Tartomány váltást* a fájl rendszeren keresztül lehet végrehajtani.
 - Minden állományhoz tartozik egy tartomány-bit (setuid bit).
 - Ha az állomány végrehajtásra kerül és a setuid bit 1, akkor a USER-ID beállítódik a végrehajtás alatt levő fájl tulajdonosának megfelelően. Ha a végrehajtás befejeződött, a USER-ID visszaáll eredeti állapotába.

- **Multics gyűrűk**

- Legyen D_i és D_j két tartomány gyűrű.
- Ha $j < i$, akkor $D_i \subseteq D_j$.



• Hozzáférési mátrixok

- Sorok: tartományok
- Oszlopok: tartományok + objektumok
- Bejegyzések: hozzáférési jogok, operátor nevek

objektum tartomány	FILE1	FILE2	FILE3	PRINTER
D1	read		read	
D2				print
D3		read	execute	
D4	read, write		read, write	

A hozzáférési mátrix használata

- Ha egy processzus a **Di** tartományban az **Oj** objektumon az “**op**” operációt kívánja végrehajtani, akkor “**op**” elő kell forduljon a hozzáférési mátrix megfelelő pozíciójában
- Kiterjeszthető dinamikus védelemmé
 - Új operációk: Hozzáférési jogok hozzáadása, törlése
 - Speciális hozzáférési jogok:
 - *owner* of **Oi**,
 - *copy* “**op**” **Oi**-ből **Oj**-be,
 - *control*: átkapcsolás a **Di** és **Dj** tartományok között
- A hozzáférési mátrix világosan elválasztja a *mechanizmust* a *politikától*
 - **Mechanizmus**: az operációs rendszer szolgáltatja a hozzáférési mátrixot + megfelelő szabályokat.
 - Biztosítja, hogy a mátrix csak arra jogosult agensek által manipulálható és a szabályokat szigorúan betartatja.
 - **Politika**: a felhasználó diktálja.
 - Ki érhet el egyes objektumokat és milyen módon teheti ezt.

- A hozzáférési mátrixok implementációja

- **Minden oszlop:** = egy **ACL** (Access Control List, hozzáférési lista) *egyetlen* objektumhoz.

Definiálja, hogy **ki** és **mit** "operálhat" az objektumon.

- **Minden sor:** = egy **CL** (Capability List, jogosultsági lista) *egyetlen* tartományhoz.

Definiálja, hogy **a különböző objektumokon mit** "operálhat" a tartományban levő processzus.

- **Lock-key rendszer** = egy objektumhoz zárok tartozhatnak, a tartományhoz kulcsok (bit pattern). Az operációs rendszer menedzseli.

• A hozzáférési jogok visszavonása

- azonnal – késleltetve
- szelektíven – általában
- részben – teljesen
- ideiglenesen – permanensen

• Megoldások:

- újrakérés
- back-pointerek
- indirekció
- kulcsok

- Képesség-alapú rendszerek
 - HYDRA
 - Cambridge CAP

- Nyelvbe ágyazott védelem
- JAVA Sandbox

14. Biztonság¹

- A biztonsági probléma
- Program fenyegetettségek
- Rendszer és hálózati fenyegetettségek
- Kriptográfia mint biztonsági eszköz
- Felhasználó azonosítás (autentikáció)
- Biztonsági védelem implementációi
- Tűzfal védelem a rendszer és hálózat számára
- Számítógép biztonsági osztályok
- Windows XP példa

¹ Silbershatz, Galvin & Gagne, Operating system concepts with Java, 7th edition (2007), John Wiley & Sons, Inc., alapján

• A biztonsági probléma

- A biztonsági probléma a rendszer erőforrásainak védelmét jelenti a rendszer külső környezetével szemben.
- Behatolók (támadók, „cracker”-ek, megkísérelhetik megsérteni a biztonságot.
- „**Fenyegetettség**” (Threat) a biztonság egy potenciális megsértése.
- „**Támadás**” (Attack) a biztonság megsértésének konkrét kísérlete.
 - A „**támadás**” lehet *véletlen*, vagy *rosszindulatú*.
 - A véletlen támadással szemben egyszerűbb védekezni, mint a rosszindulatúval szemben!

• Biztonsági sérelmek (violations)

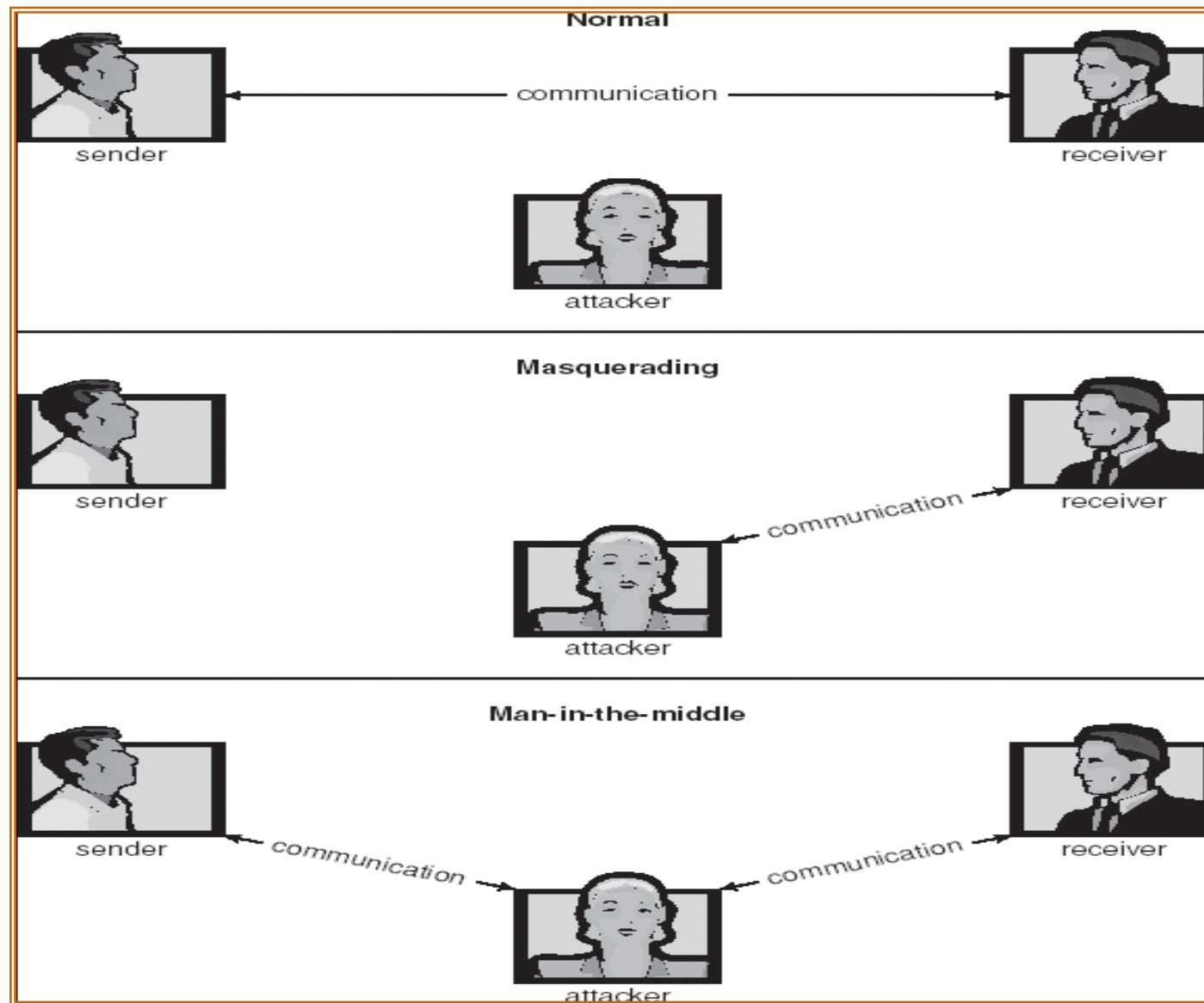
• Kategóriák

- A titkosság (confidentiality) megsértése.
- Az épség (integrity) megsértése.
- A hozzáférhetőség (availability) megsértése.
- A szolgáltatás eltulajdonítása (lopás, theft).
- A szolgáltatás megtagadása (denial of service).

• Módszerek

- Álcázás (masquerading), az autentikáció megsértése.
- Újrajátszási (replay) támadás.
- Üzenet módosítás.
- „Ügynök közbeiktatása” (Man-in-the-middle) támadás.
- Szekció rablás (Session hijacking)

• Standard biztonsági támadások



- **Biztonsági intézkedések szintjei**
- A hatékonysághoz az intézkedéseknek négy szinten kell megjeleníteniük:
 - Fizikai szint
 - Humán szint
 - Pl. elkerülni: **social engineering, phishing (whaling!), dumpster diving**
 - Operációs rendszer
 - Hálózat
- „A hálózat olyan gyenge, mint a leggyengébb láncszeme”

• Program fenyegetettségek

• Trójai faló (Trojan Horse)

- Kód szegmens (program), amelyik visszaél futtatási környezetével.
- Más nevében hajt végre olyan akciókat, amelyekre az igénybe vett felhasználó jogosult.
- Példák: Spyware, pop-up browser windows, rejtett (covert) csatornák (channels).

• Csapda (Trap Door)

- Speciális azonosító, vagy jelszó, amelyek lehetővé teszik bizonyos biztonsági ellenőrzések megkerülését.
- Beépítkező a compilerbe is!

• Logikai bomba

- Egy program, amelyik bizonyos körülmények között biztonsági problémát (security incident) vált ki.

• Verem és puffer túlcsordulás

- A program egy hibáját kihasználva a verem, illetve a pufferek túlcsordulására alapoz (, amivel a program kódja/adatai módosíthatók!).

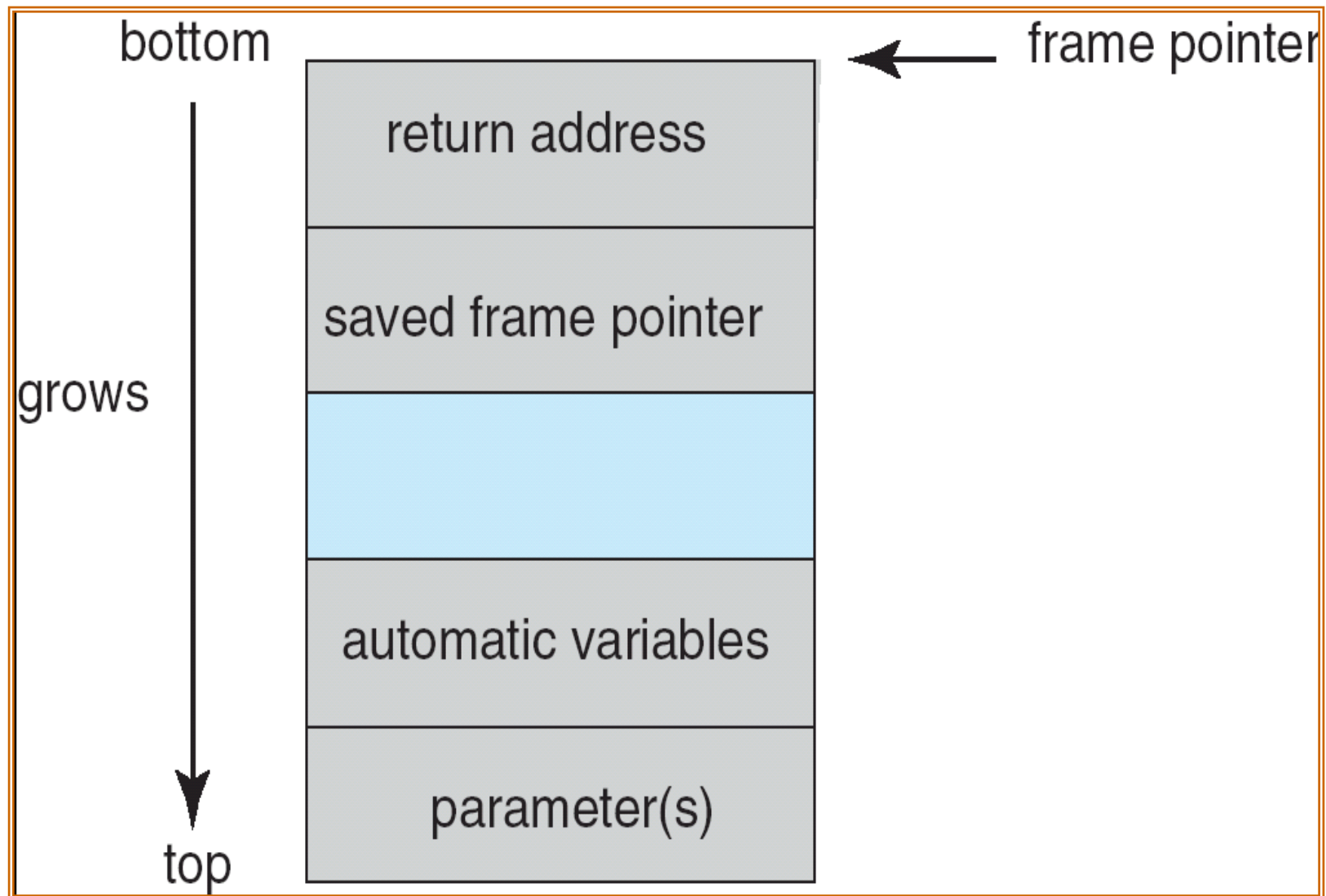
- C-program puffer túlcsordulás lehetőséggel

```
#include <stdio.h>
#define BUFFER_SIZE 256

int main(int argc, char *argv[])
{
    char buffer[BUFFER_SIZE];

    if (argc < 2)
        return -1;
    else {
        strcpy(buffer, argv[1]);
        return 0;
    }
}
```

• Tipikus verem elrendezés (layout)

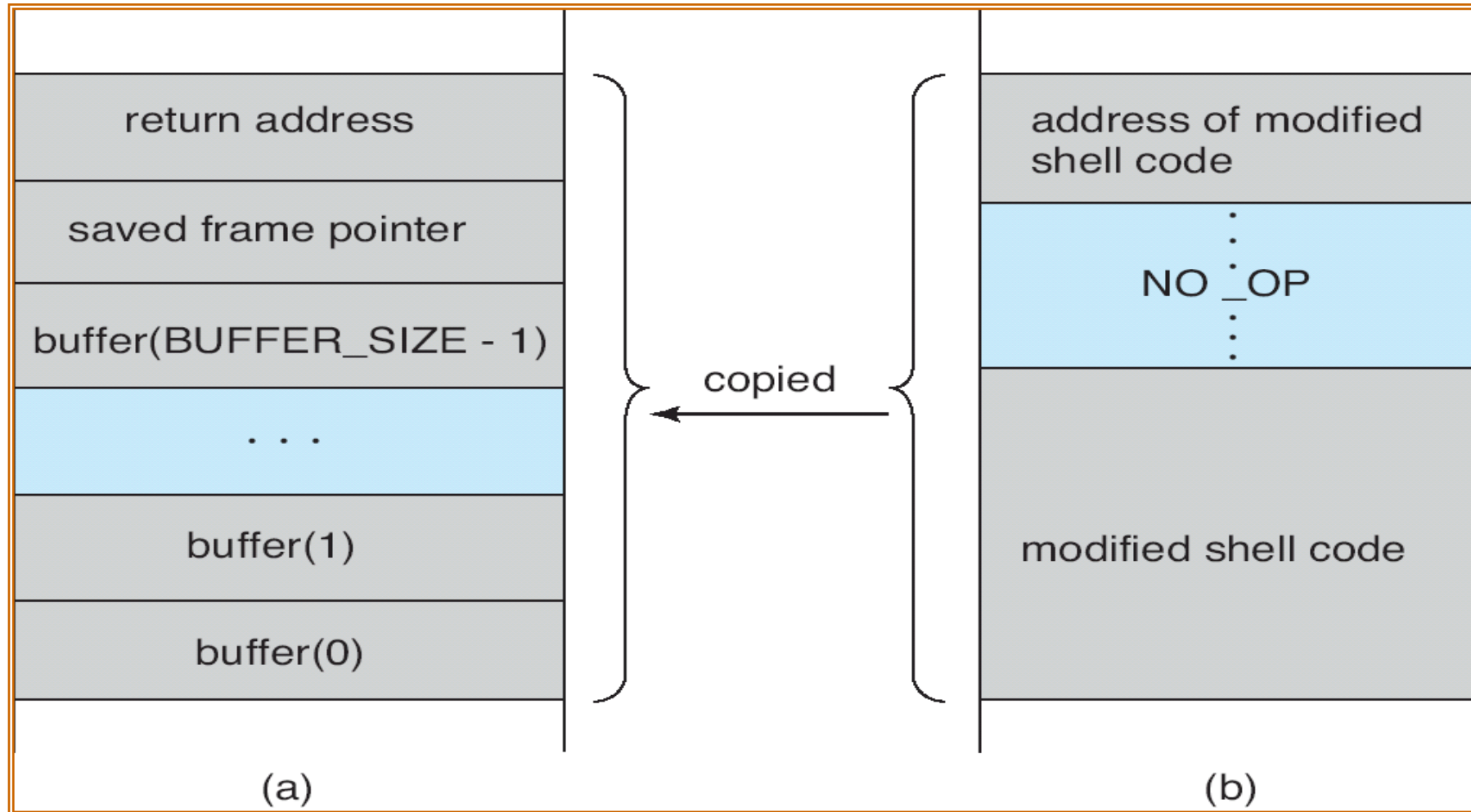


• Módosított shell kód

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    execvp(“\bin\sh”, “\bin \sh”, NULL);
    return 0;
}
```

• Hipotetikus stack keret (frame)



• Program fenyegetettségek (folyt)

• Vírusok

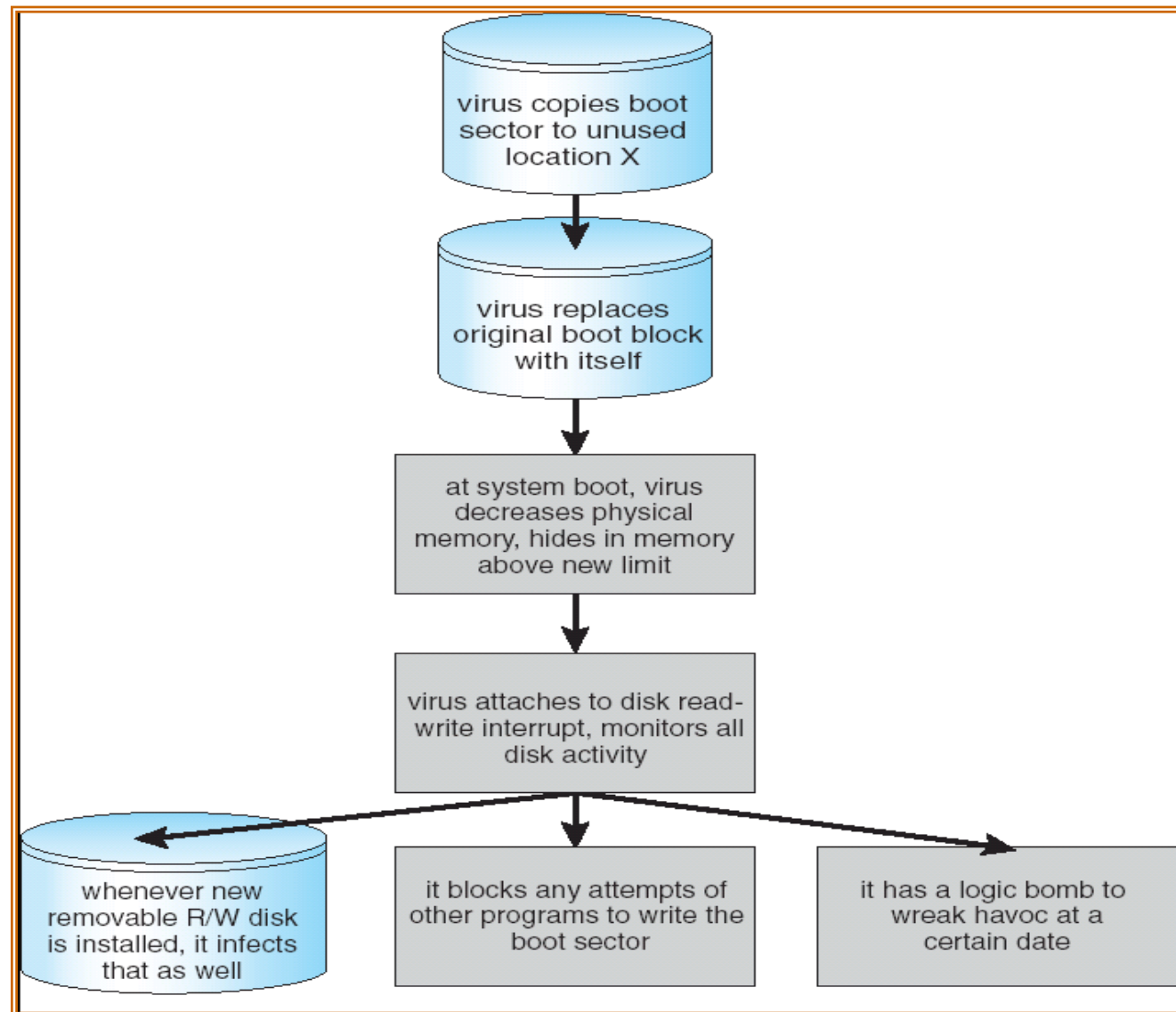
- Kódszegmens egy legitimált programba ágyazva.
- Specializált a CPU architektúrára, operációs rendszerre, alkalmazói programra.
- Rendszerint e-mail, vagy makró útján (részeként) terjed.
 - Pl.: egy Visual Basic makró, amelyik újraformázza a merevlemezt:

```
Sub AutoOpen()  
Dim oFS  
    Set oFS = CreateObject(''Scripting.FileSystemObject'')  
    vs = Shell(''c:command.com /k format    c:''',vbHide)  
End Sub
```

• Program fenyegetettségek (folyt)

- Vírus „csepegtető” (virus dropper) illeszti be a vírust a rendszerbe.
- Számos kategória, összességében több ezer (konkrét) vírus létezik!
 - File
 - Boot
 - Macro
 - Source code (forráskód)
 - Polymorphic (polimorf, több alakú)
 - Encrypted (rejtjelezett)
 - Stealth (lopakodó)
 - Tunneling (alagút)
 - Multipartite (több részre osztott)
 - Armored (páncélozott)

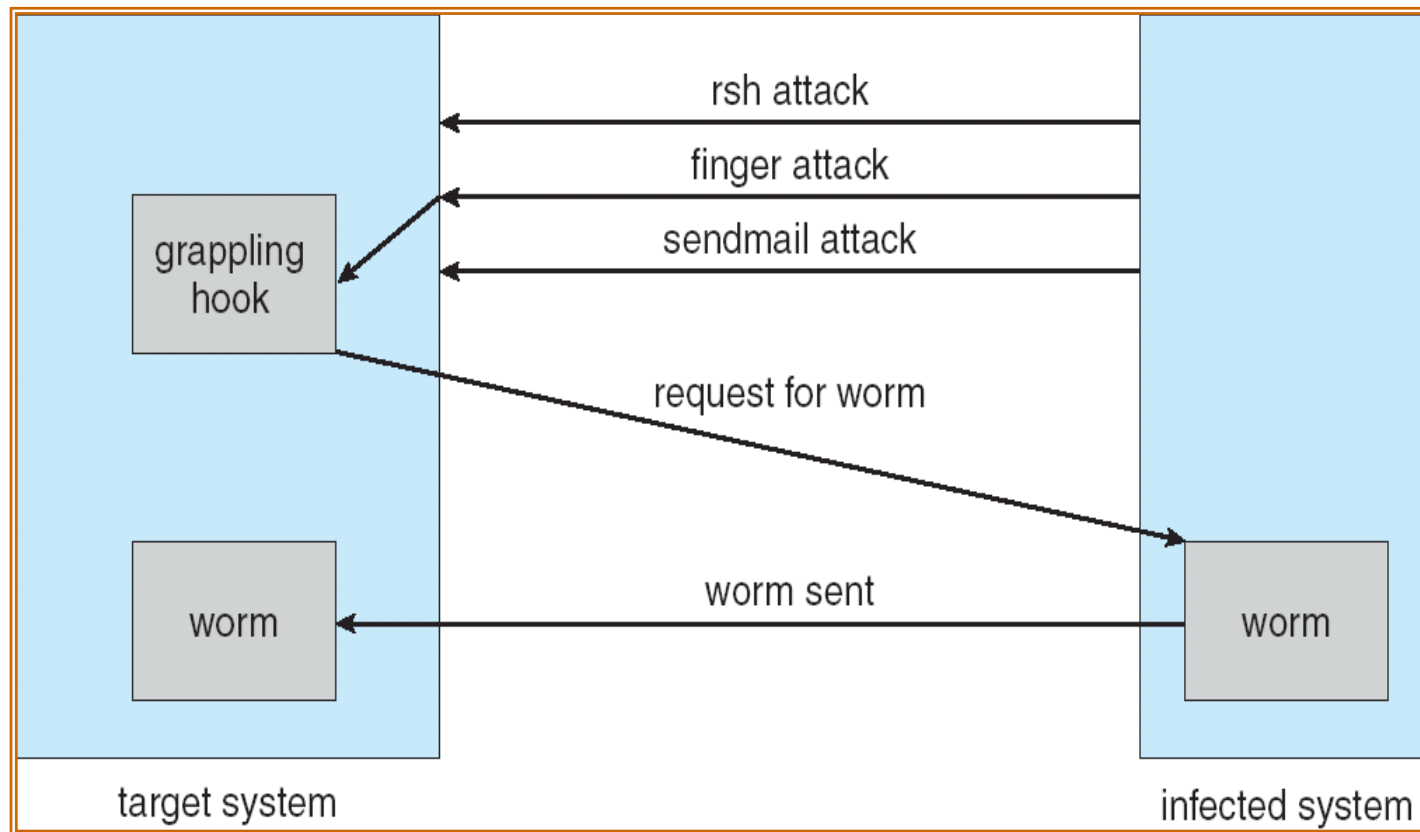
• Boot szektor vírus



- **Rendszer és hálózati fenyegetettségek**
- **Férgek** (worms) önálló programok, amelyek valamilyen szaporodási (spawn!) mechanizmust használnak a terjedéshez
- **Internet féreg**
 - A UNIX hálózati szolgáltatásait (remote access, rpc), illetve a *finger* és *sendmail* programokban levő hibákat (bug) használják ki.
 - A férget egy “horgony program” tölti le a hálózaton keresztül.
- **Port pásztázás (scanning)**
 - Automatizált kísérletek meghatározott tartományba eső IP címekhez tartozó meghatározott tartományba eső portokra történő csatlakozásra.
- **A szolgáltatás megtagadása (Denial of Service)**
 - A megcélzott számítógép (szerver) túlterhelése, hogy ne tudjon semmi „hasznos dolgot” csinálni.
 - Még veszélyesebb: Distributed denial-of-service (DDOS), ahol sokan támadnak egyszerre.

• A Morris – féle Internet féreg

- 1988. november 2.: Robert Tappan Morris (első éves Cornell egyetemista)
- SUN3, VAX BSD Unix.
- 3 év próba, 400 óra közmunka, 10000\$ pénzbüntetés



- Más: 2003. augusztus: Sobig.F

• Port scanning Java-ban

```
public class PortScanner
{
    public static final int PORT_MAX = 255;

    public static final int TIMEOUT_VALUE = 1000; // 1 second

    public static void main(String[] args) {
        InetAddress host = InetAddress.getByName(args[0]);

        for (int port = 0; port <= PORT_MAX; port++) {
            try {
                SocketAddress addr = new InetSocketAddress(host, port);
                Socket sock = new Socket();

                // attempt to make a connection to (host + port)
                sock.connect(addr, 1000);
                System.out.println("Listening at port: " + port);
                // we could now try to exploit the service
                // listening at this port
            }
            catch (java.io.IOException ioe) {
                // not listening to this port
            }
        }
    }
}
```

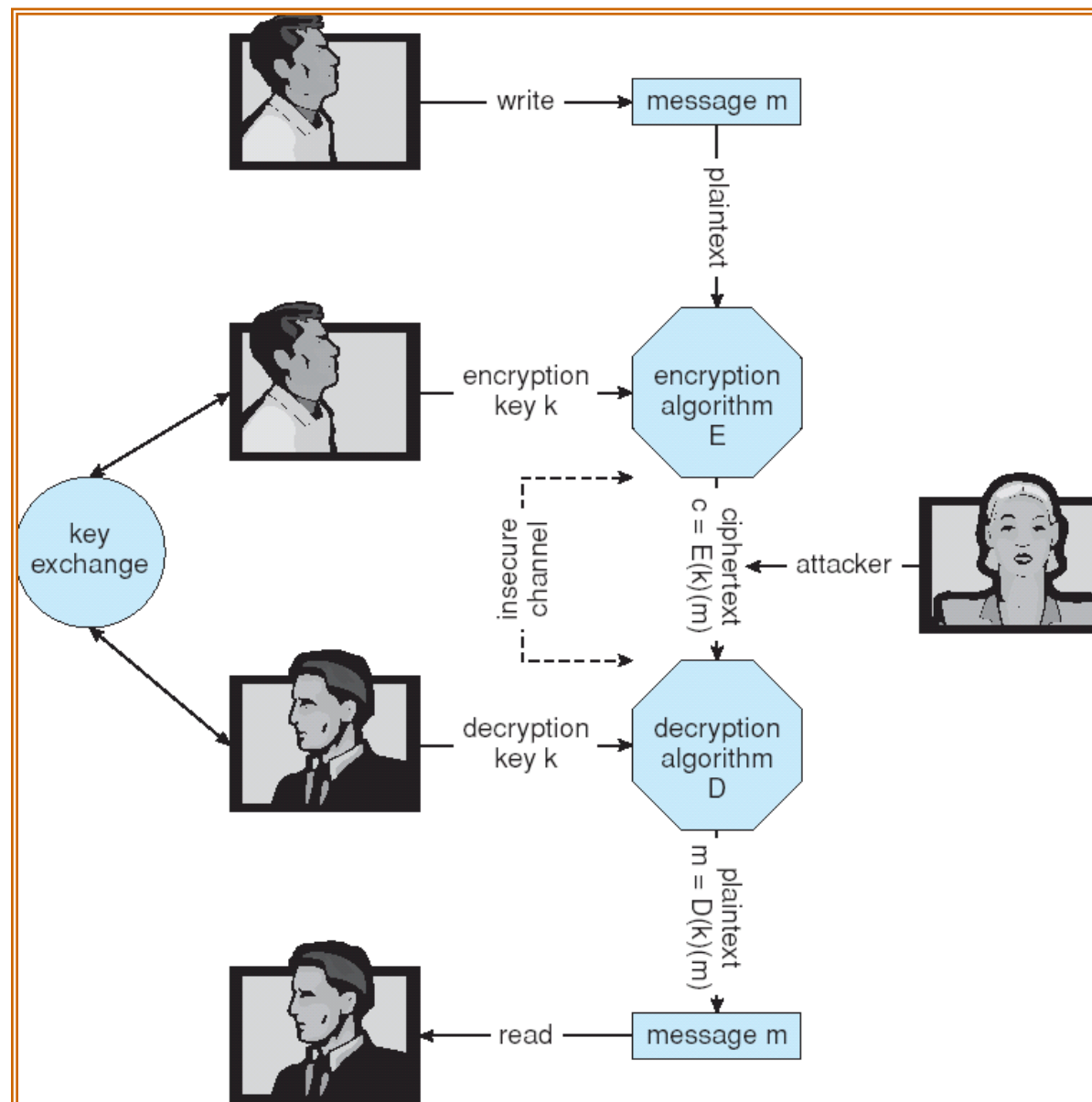

• **Kriptográfia mint biztonsági eszköz**

• **Az elérhető legáltalánosabb biztonsági eszköz**

- Az üzenet forrása és célállomása nem lehet bizalomra méltó kriptográfia nélkül.
- Eszköz a potenciális adók (források) és/vagy a vevők (célállomások) korlátozására.

• **Valamilyen titkok (kulcsok) képezik a módszerek alapját**

• Biztonságos kommunikáció közönséges eszközökkel



• Rejtjelezés (kódolás, encryption)

• A rejtjelező algoritmus elemei

- K a kulcsok halmaza (keys)
- M az üzenetek halmaza (messages)
- C a rejtjelezett szövegek halmaza (cipher texts)
- Egy függvény $E : K \rightarrow (M \rightarrow C)$.
 - Azaz, minden $k \in K$, $E(k)$ egy olyan függvény amely üzenetekhez rejtjelezett szöveget rendel.
 - Mind E és $E(k)$ (minden lehetséges k -ra) hatékonyan kiszámítható függvény kell legyen.
- Egy függvény $D : K \rightarrow (C \rightarrow M)$.
 - Azaz, minden $k \in K$, $D(k)$ egy olyan függvény amely rejtjelezett szövegekhez üzeneteket rendel.
 - Mind D és $D(k)$ (minden lehetséges k -ra) hatékonyan kiszámítható függvény kell legyen.

• Aszimmetrikus rejtjelezés

- A nyilvános kulcsú rejtjelezés esetén minden partner két kulccsal rendelkezik:
 - **Nyilvános kulcs** (public key), amely publikus és a rejtjelezéshez használatos.
 - **Titkos kulcs** (private key), amelyet minden partner titokban tart és a rejtjelezett szöveg visszaalakításához használ.
- Szükség van egy rejtjelező sémára, amelyet nyilvánosságra lehet hozni anélkül, hogy belőle a visszafejtő sémát könnyen ki lehetne találni.
- Pl. egy ilyen rendszer az RSA
- 1978. R. Rivest, A. Shamir, L. Adleman (Tel Aviv University)
- gyors prímtesztek,
- lassú faktORIZÁCIÓ!

• Aszimmetrikus rejtjelezés (folytatás)

- Összefoglalás:
- Az RSA esetében reálisan lehetetlen meghatározni a $D(k_d, N)$ -t az $E(k_e, N)$ -ből, $E(k_e, N)$ -t nem szükséges titokban tartani és ezért széles körben terjeszthető
 - $E(k_e, N)$ (vagy k_e) a nyilvános kulcs (**public key**)
 - $D(k_d, N)$ (vagy k_d) a titkos (magán) kulcs (**private key**)
- Emlékeztetőül: N két nagy véletlen módon választott prím szám p és q szorzata (pl. p és q 512 bit méretűek)
- **Titkosítási algoritmus:**

$$E(k_e, N)(m) = m^{k_e} \bmod N,$$

ahol k_e eleget tesz a $k_e k_d \bmod (p-1)(q-1) = 1$ kongruenciának.

- **A visszafejtő algoritmus ekkor**

$$D(k_d, N)(c) = c^{k_d} \bmod N.$$

• Aszimmetrikus rejtjelezés (egyszerű! példa)

Legyen $p = 7$ és $q = 13$.

Akkor kiszámítható, hogy $N = 7 * 13 = 91$ és $(p-1)(q-1) = 6 * 12 = 72$.

Választunk egy k_e 72-höz relatív prím és < 72 számot, mondjuk legyen ez 5.

Végül meghatározzuk k_d -t úgy, hogy

$$k_e k_d \equiv 1 \pmod{72}$$

teljesüljön, ami, $k_d = 29$ -hez vezet.

Kulcsaink:

Publikus kulcs: $k_e, N = 5, 91$

Privát kulcs: $k_d, N = 29, 91$

Kódolás–dekódolás:

A “69” üzenet kódja a nyilvános kulcs segítségével: $69^5 \equiv 62 \pmod{91}$.

A titkos üzenet (“62”) dekódolható a titkos kulcs segítségével: $62^{29} \equiv 69 \pmod{91}$.

• Autentikáció

- Constraining set of potential senders of a message
 - Complementary and sometimes redundant to encryption
 - Also can prove message unmodified
- Algorithm components
 - A set K of keys
 - A set M of messages
 - A set A of authenticators
 - A function $S : K \rightarrow (M \rightarrow A)$
 - That is, for each $k \in K$, $S(k)$ is a function for generating authenticators from messages
 - Both S and $S(k)$ for any k should be efficiently computable functions
 - A function $V : K \rightarrow (M \times A \rightarrow \{\text{true}, \text{false}\})$. That is, for each $k \in K$, $V(k)$ is a function for verifying authenticators on messages
 - Both V and $V(k)$ for any k should be efficiently computable functions

• Autentikáció

- For a message m , a computer can generate an authenticator $a \in A$ such that $V(k)(m, a) = \text{true}$ only if it possesses $S(k)$
- Thus, computer holding $S(k)$ can generate authenticators on messages so that any other computer possessing $V(k)$ can verify them
- Computer not holding $S(k)$ cannot generate authenticators on messages that can be verified using $V(k)$
- Since authenticators are generally exposed (for example, they are sent on the network with the messages themselves), it must not be feasible to derive $S(k)$ from the authenticators