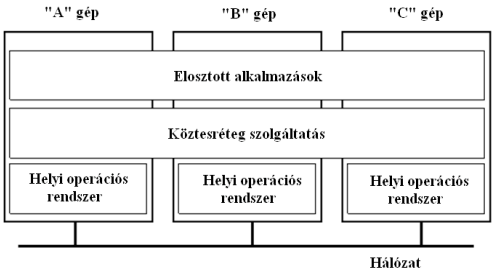
27 - Osztott rendszerek

- Def: Az elosztott rendszer az önálló számítógépek olyan összessége, amely kezelői számára egyetlen koherens rendszernek tűnik.
- Hátrányok: hatékonyság csökkenése, gyengébb biztonság | Fajtái:



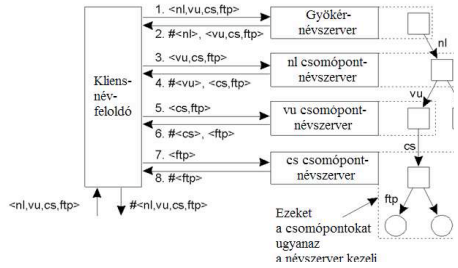
1. Elosztott oprendszerek: több processzoros és homogén multi-számítógépek szorosan csatolt operációsrendszere | Hardver erőforrások kezelése, elrejtése
2. Hálózati oprendszerek: Heterogén multi-számítógépek (LAN,WAN) gyengén csatolt operációsrendszere | Helyi szolgáltatások távoli kliensek számára elérhetővé teszi
3. Köztesrétegként felépülő elosztott rendszer (megvalósítja az ER definícióját) →

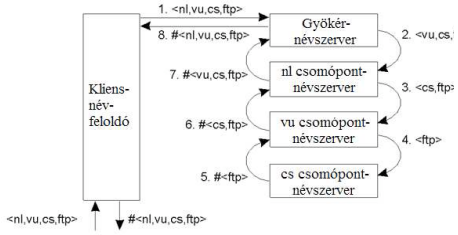
I Köztesrétegek tulajdonságai: Elrejti az általa használt platformok különbözőségét
1. Felhasználók és erőforrások összekapcsolása: távoli erőforrások elérése (Pl. nyomtató)
2. Átlátszóság: Átlátszó, ha egy számítógépből álló rendszernek tűnik a felhasználói számára
3. Nyitottság: Nyitott, ha az általa nyújtott szolgáltatásokat jól definiált szintaktikai és szemantikai szabályoknak megfelelően biztosítja | Jellemzői:
- Együttműködés (különböző cégek által kifejlesztett rendszerek között)
- Hordozhatóság (különböző platformok között)
- Megvalósítás és módszer szétválasztása (kicsi, könnyen cserélhető részek)

4. Átméretezhetőség: Nagyságában, földrajzi átméretezhetőségében és adminisztrálhatóságában | Problémák:
- **Központosított szolgáltatás** (egy kiszolgáló minden felhasználónak, bizt-sabb)
- **Központosított adat** (egy darab adatbázis) → Megoldás: DNS
- **Központosított algoritmus** (tejes infón alapuló útválasztás)
→ Megoldás: Decentralizált algoritmus

Hozzáférhetőségi	Elrejti az adatábrázolási különbségeket
Elhelyezkedési	Elrejti az erőforrás rendszeren belüli fizikai elhelyezkedését
Áthelyezhetőségi	Elérésük nem változik meg az erőforrások áthelyezése során
Mozgathatósági	Erőforrás áthelyezhető használat közben,erről a felhasználó nem szerez tudomást
Többszörözhetőségi	Teljesítmény növelése miatt
Egyidejűség	Egyik felhasználó sem veszi észre, hogy rajta kívül más is használja az erőforrást
Meghibásodási	Felhasználó nem veszi észre ha valamelyik erőforrás nem működik megfelelően
Állandóság	Elrejti, hogy az erőforrás a memóriában vagy a háttértárolón van-e

II Elnevezési rendszerek:
- **Név**: ER-ben a karakterek vagy bitek olyan sorozata, amely valamilyen entitás azonosítására szolgál (entitás lehet gazdagép, nyomtató, folyamat …)
3féle név: **Cím**: entitás elérésére használt kapcsolódási pont neve | **Azonosító**: minden entitásra pontosan egy azonosító hivatkozik, egy azonosító csak egy entitásra hivatkozik, az egyszer kiosztott azonosító soha nem fog másik entitásra hivatkozni | **Emberközeli nevek**: általában karakter sorozat
Globális név: rendszerben bárhol használható | **Helyi név**: értelmezése helyfüggő
- **Névtér**: a neveket névterekbe szervezzük | a névtér elnevezési gráffal írható le (gyökérpontos, címkézett, irányított) | gyökér, levél, katalógus csomópont | Entitásra útvonal névvel lehet hivatkozni
- **Névfeloldás**: gráf bejárása úgy, hogy az útvonal összetevőit egymásután keressük meg (egyszerre csak egyet) | iteratív és rekurzív módon történhet →

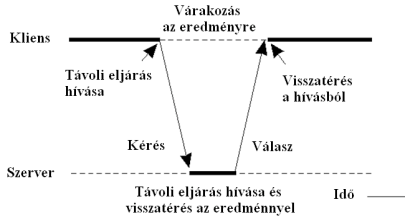




Mozgó entitás helyének meghatározását sokkal hatékonyabb helyfüggetlen azonosítók segítségével végezni, ezeknek 4 fajtája van:
1. Adatszórás/csoportcímzés: entitás azonosítóját elküldjük az ER minden folyamatának | csak az adott entitáshoz tartozó kapcsolódási pontot biztosító folyamat válaszol | Probléma: átméretezhetőség
2. Továbbító mutató: amikor egy entitás elköltözik, régi címén meghagyja az új címére hivatkozó mutatót | helymeghatározáskor a mutatók láncán végighaladunk | Probléma: túl hosszú lánc
3. Otthon alapú módszer: az entitáshoz otthont rendelünk | valahányszor elköltözik új címét megadja az otthonnak
4. Hierarchikus módszer: Hierarchikus keresőfát építünk | legfelső szintű tartomány valamennyi entitást ismeri | legalacsonyabb szintű tartomány tárolja az entitás címét

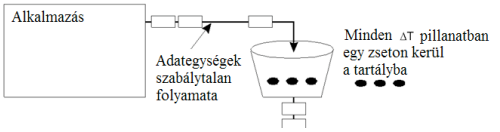
Nem hivatkozott entitások eltávolítása:
- Cél: Entitások, amelyek helye nem határozható meg, könnyen törölhetők legyenek (Szemétgyűjtés)
1. Hivatkozás számlálás: Megszámolja hány külső hivatkozás mutat rá a rendszerben | amikor ez a szám 0, az entitás törölhető | Prob: átméretezhetőség
2. Nyomkövetés alapú módszer: Szükséges hogy legyen hivatkozás a gyökérkészletből | Ha van elérhetőnek jelöljük | Jelöletleneket töröljük

III Kommunikáció:
1. Távoli eljárás hívás (Remote Procedure Call): Elrejti a kommunikáció tényét | Hozzáférési átlátszóság
- Szolgáltatásokat a szervereken futó eljárások formájában implementáljuk
- Kliensnek csak az eljárás neve és paraméterlistája áll rendelkezésre
- Valahányszor a kliens az adott eljárást hívja, egy csonknak nevezett kliensoldali implementáció csomagolja üzenetbe a paramétereket, majd az üzenetet elküldi a szervernek
→ A szerver meghívja a kívánt eljárást, majd visszatér az eredménnyel, üzenet formájában
→ Kliensoldali csonk az eredményt kicsomagolja a visszatérő üzenetből és visszaadja a kliens alkalmazásnak | szinkron RPC →



2. Távoli metódus hívása (Remote Method Invocation): RPC ötletét kiterjesztjük távoli objektumok hívására
Elosztottsági átlátszóságot jobban megvalósítja | Támogatja a rendszerszintű objektumhivatkozások paraméterként való átadását
3. Üzenetorientált kommunikáció: RPC, RMI nem mindig használható | szerver nem biztos, hogy üzemel a kérés elküldésekor (Pl. AB kezelés, email)
- Megtartó: a rendszer addig tárolja az üzenetet, amíg el nem jut a címzetthez
- Időleges: a rendszer csak akkor tárolja az üzenetet ha a küldő és a fogadó alkalmazás is fut
- Szinkron: blokkolódik a küldő amíg az üzenet meg nem érkezik/amíg a fogadó fel nem dolgozza
- Aszinkron: a feladó az üzenet elküldése után azonnal folytatja futását

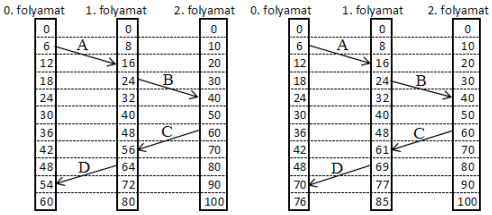
4. Adatfolyamok: Időfüggő információk cseréje
- Diszkrét ábrázolású média (szöveg, kép) | Folyamatos ábrázolású média (hang, videó) |
- Zsetontartály-algoritmus hatására az adat viszonylag állandó időközönként jut át a hálózaton →



IV Szinkronizáció:

- Probléma: Nincs globálisan elérhető óra | Különböző gépeken futó folyamatoknak saját elképzelésük van a pontos időről

Óraszinkronizáló algoritmusok: (fizikai órák esetén)



1. **Cristian algoritmusa**: Pontos idő lekérdezése az időszerverről | Bele kell számítani a válaszidőt is
   Ha időt vissza kell állítani, akkor azt fokozatosan teszik | másodpercenként H megszakítás
2. **Berkeley algoritmus**: Itt az időszerver aktív | Időnként lekérdezi az összes gépet
   Egyes gépek saját óráját gyorsításra, illetve lassításra kényszeríti
3. **Átlagoló algoritmus**: Decentralizált algoritmus | Minden gép kihirdeti a saját idejét | Átlagol
4. **Többszörözött külső időforrások**: Több UTC (egyetemesen szabályozott idő)
   Ezek rendszeresen kihirdetik az idejüket

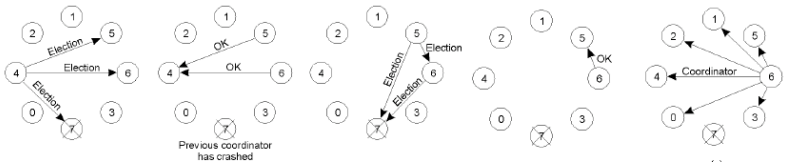
Lamport időbélyege: (logikai óra, melynél az események sorrendje a fontos) →

- 3 folyamat | mindegyik saját órával | az órák különböző sebességgel futnak

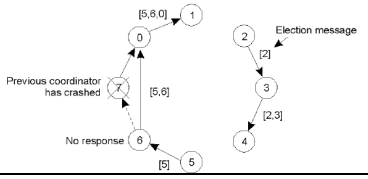
Időbélyeg vektor: Oksági viszony alkalmazása: Ha VT(a) < VT(b), akkor a b-nek előzménye

Szavazó algoritmusok: - Kitüntetett, koordinátor folyamat választása, a folyamatok egyedi azonosítójának a segítségével
- Legnagyobb/legkisebb azonosítójú folyamat megtalálása | Egyetértés biztosítása

1. Zsarnok: Ha egy folyamat szavazást indít a nála magasabb sorszámúaknak üzen
Ha senki nem válaszol ő nyeri a szavazást | Egyébként a válaszoló folyamatok folytatják a szavazást, amíg a legnagyobb érétkű folyamat nem nyer

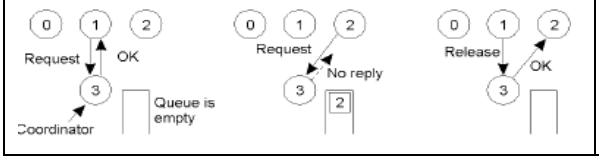


2. Gyűrű: Folyamatok sorba vannak rendezve
Ha egy folyamat észreveszi, hogy a koordinátor nem üzemel, kezdeményez egy szavazást | Üzenet végighalad a folyamatokon
Minden folyamat hozzáadja a sorszámát

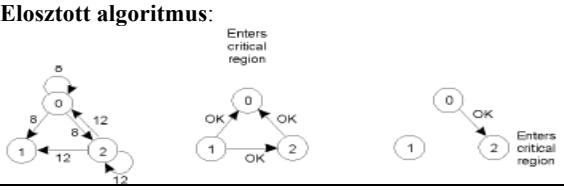


3. Kölcsönös kizárást biztosító algoritmusok: valamilyen erőforrás megszerzése kizárólagos használatra

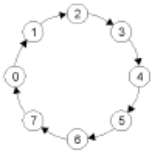
Központosított algoritmus:



Elosztott algoritmus:



Zsetongyűrű algoritmus:



Elosztott tranzakció: Védelem a konkurens hozzáférés ellen
Tranzakciók tulajdonságai (**ACID**): Oszthatatlan (Atomic) | Konzisztens (Consistent) | Elkülönülő (Isolated, sorosítható) | Maradandó (Durable)

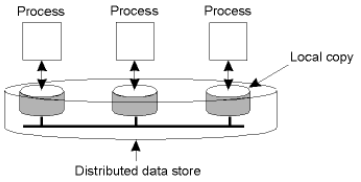
V Konzisztencia:

- Többszörözés okai: Rendszer teljesítményének és a megbízhatóság növelése
- Probléma: Adatmásolatok konzisztenciájának a biztosítása (egy másolat frissítésekor a többi másolatot is frissítenünk kell)
- Konzisztencia modell: Olyan szabványrendszer, amit ha a folyamatok betartanak, akkor az az adattár helyes működését eredményezi
  (adattár lehet egy közös adatbázis vagy állomány rendszer)

1. Adatközpontú modellek: Fizikailag több folyamat között elosztott és többszörözött logikai adattár általános felépítése

Szinkronizáló műveletet nem igénylő konzisztencia modellek:

Szigorú	Olvasási művelet mindig a legutoljára írt értékkel tér vissza
Soros	Írási műveleteket mindenki ugyanabban a sorrendben látja (nincs idő szerint sorba rendezve)
Lineáris	= *Soros* + műveleteket globális óra segítségével állítja sorba
Okozati	Egymástól okozatilag függő műveletek a függőség sorrendjében következnek
FIFO	Valamennyi folyamat a többi írási műveleteit pontosan a kiadás sorrendjében fogja látni. A különböző folyamatok által végrehajtott írási műveletek sorrendje bármi lehet.



Szinkronizáló műveleteken alapuló konzisztencia modellek:

Gyenge	A megosztott adat konzisztens voltára csak a szinkronizálás végrehajtása után lehet számítani.
Feloldó	A megosztott adat konzisztensé válik a kritikus terület elhagyásakor.
Belépő	A kritikus területhez tartozó megosztott adat konzisztensé válik a kritikus területre való belépéskor.

2. Kliensközpontú modellek: Adattárak különleges csoportja: Nincs párhuzamos módosítás | Legtöbb művelet csak olvassa az adattárat

- Egyszerűbb modell, könnyebb implementálni
- Kliensközpontú konzisztencia: Egyetlen kliens szemszögéből ügyel a konzisztenciára (vándorló felhasználók számára alakították ki)
- Feltételezés, hogy a kliensek az idő múlásával különböző másolatokhoz csatlakoznak → ennek átlátszó módon kell történnie
- Másolat arra az állapotra frissítődik, amelyen a kliens korábban dolgozott