# Beágyazott információs rendszerek

**4. Időmérés, időszolgáltatás, óra-szinkronizáció**

2020. október 15.

Emlékeztető

***A fizikai referencia óra:*** jele $\boldsymbol{C}$, felbontóképessége $g^C$. Pl.: $10^{15}$ óraütés/sec → $g^C = 10^{-15}$ sec. Értéke a nemzetközi idő szabvány szerinti abszolút idő.

***Időbélyeg:*** $\boldsymbol{C}(e)$: az *e* esemény abszolút időbélyege.

***Óra drift:*** *A k*-jelű fizikai óra két, önkényesen kiválasztott óraütése között eltelt időt a referencia órával megmérjük, és a vizsgált óra által mutatott időkülönbséget viszonyítjuk ennek teljesen pontos értékéhez:

$$drift_k(t_i, t_{i+1}) = \frac{C_k(t_{i+1}) - C_k(t_i)}{C(t_{i+1}) - C(t_i)}$$

Mivel a ***drift*** ideális értéke 1, ezért szokás definiálni a ***drift-mértéket***: $\delta_k = \rho_k = |drift_k - 1|$ formában, ami specifikációs adat az órára, egyhez képest nagyon kis érték ($10^{-2}$...$10^{-7}$ sec/sec). Előfordul, hogy a szóhasználat ezt nevezi ***drift***-nek, ami a nagyságrendi eltérés miatt nem okoz félreértést. A ***drift mérték*** maga a ***drift*** egytől való eltérésének előjelét nem hordozza. Ha nincs szinkronizáció, akkor az órák a ***drift*** következtében eltérő ütemben haladnak, „elmásznak”.

Ennek súlyos következményei lehetnek! **Példa:** Öböl háború, Dhahran, 1991. február 25.

Egy **Patriot** rendszer elvétett egy **Scud** rakétát.
Egy fizikai óra mintegy **100 óráig** (>4 nap) szinkronizálás nélkül maradt, ez alatt - kvantálási hiba következtében - összeszedett **0.3433 sec** késést, ami **687 méteres** követési hibát okozott a célkövető számításaiban, és ez által a mintegy **1.7 km/sec** sebességgel haladó rakéta kikerült célkövető látóköréből. Következmény: **28 halott, 98 sebesült**! A hiba hátterében az állt, hogy korábban a **Patriot** rendszereket rövidebb működési idő feltételezésével, lényegesen lassabb eszközök ellen fejlesztették, és az Öböl háború idején fejlesztették tovább **Scud** rendszerekhez.

Emlékeztető

A konkrét tragédiát okozó hibát már február elején felfedezték, február 16-án **a módosított szoftvert ki is adták**, de az nem jutott el az érintett **Patriot** rendszerbe.

***Óra ofszet:*** Tekintsünk két órát azonos felbontóképességgel: $ofszet_{j,k}(t) = \left|C_j(t) - C_k(t)\right|$

***Együttfutás* (precision):** Tekintsünk n órát! Az együttfutás: $\Pi(t) = \underbrace{max}_{\forall 1 \le j,k \le n} \left[ofszet_{j,k}(t)\right]$

A drift miatt ez az idő múlásával nő, ezért kell szinkronizálni.

Ez az ún. "**belső szinkronizáció**", mert az órákat egymáshoz igyekszünk szinkronizálni.

***Pontosság* (accuracy):** a $k$ jelű óra ofszetje a referencia órához képest:

$ofszet_{k,ref}(t) = \left|C_k(t) - C_{ref}(t)\right|$

A drift miatt ez az idő múlásával nő, ezért kell szinkronizálni.

Ez az ún. **"külső szinkronizáció",** mert az órákat a referencia órához igyekszünk szinkronizálni.

**Példa: Igaz-e a következő állítás?**

Ha minden óra a vizsgált halmazban kívülről szinkronizált ***A* pontossággal**, akkor az óra-együttes belülről is szinkronizált **2*A* együttfutással**.

Az állítás igaz!

Fordítva nyilván nem.

## Az idő mérése elosztott rendszerekben

Az eddigiektől eltérően jellegzetesen különböző órákkal; az elosztott rendszerben mindenkinek saját órája van.

**Globális idő:** az univerzális referencia idő "gyengített" változata.

Tegyük fel, hogy a csomópontokban lévő $C_k$ órák $g^k$ felbontással ketyegnek.

Belülről szinkronizáltak $\Pi$ együttfutással, azaz tetszőleges $j$ és $k$ párra $\left|C_j(t) - C_k(t)\right| < \Pi$ $\forall t$-re. A **globális idő** az univerzális referencia idővel azonos pontosságú, de durvább felbontású óraként fogható fel, melynek ütései az ún. **makro-ütések**.

Mikor használható értelmesen a globális idő?

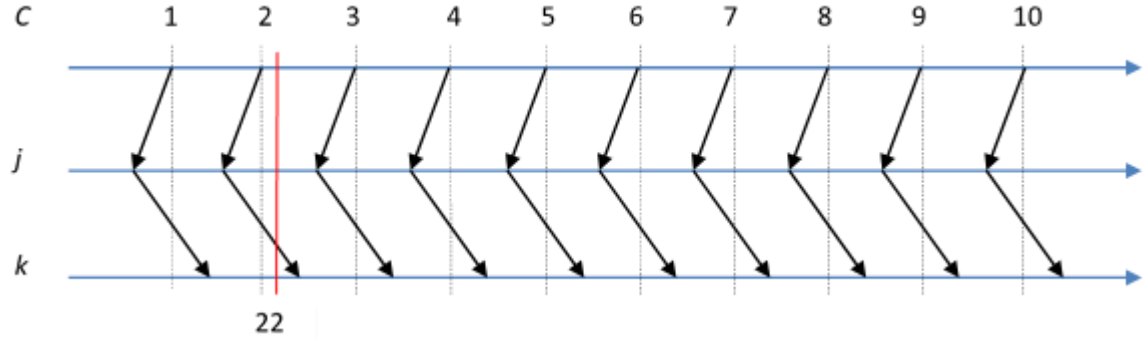
Emlékeztető

Ha a globális idő felbontása $g > \Pi$, vagyis a szinkronizációs hiba kisebb, mint a felbontó-képesség!

Ez egyben azt is jelenti, hogy egy ***e*** esemény időbélyegei a ***j*** és ***k*** csomópontok globális idő értékeivel legfeljebb egyetlen értékben különböznek.

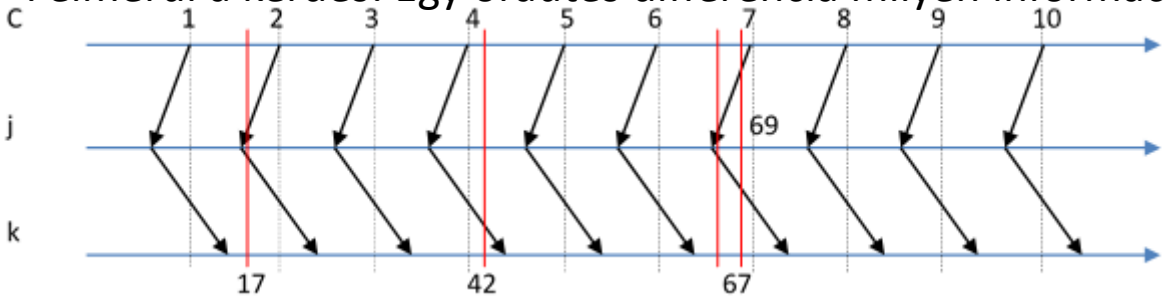
$|C_j(e) - C_k(e)| \leq 1$. Ez a legjobb, amit elérhetünk, mert mindig előfordulhat az a szituáció, hogy először a ***j*** óra üt, majd bekövetkezik az ***e*** esemény, majd üt a ***k*** óra is. Ilyenkor a két óra egy óraütés differenciával bélyegzi az ***e*** eseményt.

**Példa** (minden makro-ütés tíz mikro-ütésnek felel meg):




Az $e$: 22 mikro-ütésnél lévő eseményt $j$: 2-nek, $k$: 1-nek jelzi

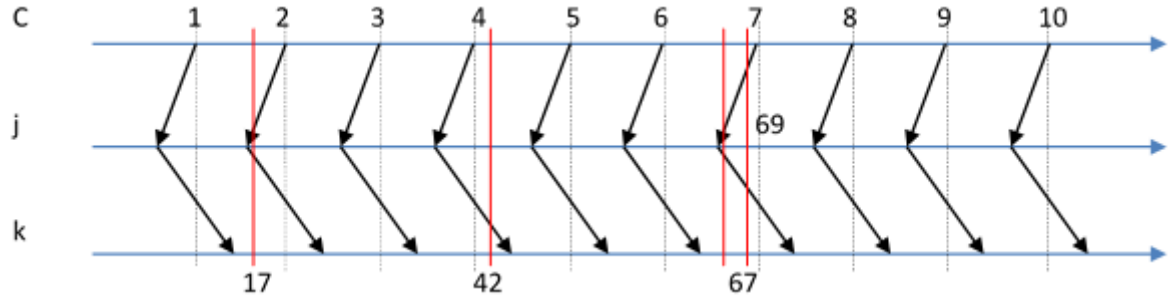
Felmerül a kérdés: Egy óraütés differencia milyen információt hordoz?




Az $e$: 17 mikro-óraütésnél $j$: 2, $k$: 1.

Az $e$: 42 mikro-óraütésnél $j$: 4, $k$: 3.

Ha az $e$: **42** és az $e$: **17** események időkülönbségét a $C_k$ és $C_j$ órák különbségeként mérjük, akkor a mérés **1**-et ad a globális időbélyegben annak ellenére, hogy a tényleges különbség **25** mikro-ütés.




Az $e$: 67 mikro-óraütésnél $j$: 7, $k$: 6.

Az $e$: 69 mikro-óraütésnél $j$: 7, $k$: 6.

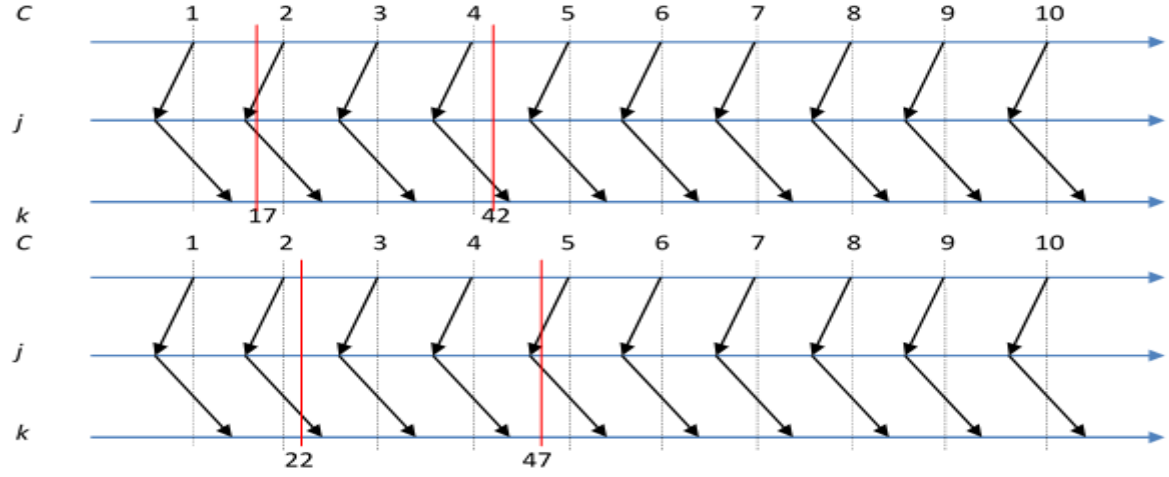
Ha az $\boldsymbol{e}$: **69** és az $\boldsymbol{e}$: **67** események időkülönbségét a $C_j$ és $C_k$ órák különbségeként mérjük, akkor a mérés **1**-et ad a globális időbélyegben annak ellenére, hogy a tényleges különbség **2** mikro-ütés.

**Probléma:** Az időbeni sorrendet a második esetben nem tudjuk egyértelműen megállapítani a makro-ütések alapján! Az $\boldsymbol{e}$: **67** mikro-óraütésnél $\boldsymbol{j}$: **7**, az $\boldsymbol{e}$: **69** mikro-óraütésnél $\boldsymbol{k}$: **6**!

**Állítás**: Ha két makro óraütés a differencia, akkor már meg tudjuk mondani az időbeni sorrendet, mert a szinkronizálási és a digitalizálási hiba mindig kisebb, mint 2.

**Idő-intervallum mérése:** $(d_m - 2g) < d_v < (d_m + 2g)$

ahol $d_v$ az intervallum tényleges értéke, $d_m$ pedig a mért érték.




Az $e$: 42 és az $e$: 17 események $C_k - C_j$ időkülönbségét mérve a mérés 1-et ad. (25 mikro-ütés.)

Az $e$: 47 és az $e$: 22 események $C_j - C_k$ időkülönbségét mérve a mérés 4-et ad.
(Ez is 25 mikro-ütés!)

Emlékeztető

## Óra rendszerek típusai:

- **Központi óra rendszerek** *(central clock systems)*:
  - egy pontos óra szolgáltatja az időt a teljes rendszer számára,
  - hibatűréshez készenléti (standby) redundanciát alkalmaznak,
  - pontos módszer (ns-en, ms-en belül), költséges
  - a kommunikációs igény alacsony (egy üzenet frissítésenként),
  - a GPS (Global Positioning System) jó példa erre (4 órajelet sugárzó műhold, ns pontosság).
- **Központilag felügyelt óra rendszerek** *(centrally controlled clock systems)*:
  - egy (pontosnak elfogadott) master óra lekérdezi a slave órákat,
  - megmérik az óra eltéréseket és a master korrekciót ír elő a slave számára,
  - ha a master óra meghibásodik, akkor (választási algoritmussal) új master-t választanak,
  - az átviteli időket és a késleltetéseket becsülni kell, mert lényegesen befolyásolják a mért óra eltéréseket,
  - a kommunikációs terhelés erősebb, mint előbb.
- **Elosztott óra rendszerek** *(distributed clock systems)*
  - az óra szempontjából az összes csomópont homogén, ugyanazt az algoritmust futtatja,
  - minden csomópont frissíti az óráját, miután megkapta, és helyesség szempontjából ellenőrizte/becsülte a más órák által kapott időt,
  - a hibatűrés protokoll alapú. Ha egy csomópont kiesik, az nem befolyásolja a többi csomópont működését; észlelik a hibát és figyelmen kívül hagyják a meghibásodott csomópontot,
  - a kommunikációs igény viszonylag nagy, különösen akkor, ha alattomos hibák (pl. bizánci hibák) esetén is a robusztusság követelmény.

## Idő normáliák (standardok)

- **Nemzetközi Atomi Idő** *(Temps Atomique Internationale, TAI)*.
  Alapja egy ún. atomóra: Cesium-133 atom által (specifikált módon) kisugárzott frekvencia 9 192 631 770-ed része 1 sec. A TAI által biztosított időskála kronoszkópikus, azaz folytonos.
- **Univerzális idő** vagy **Egyezményes koordinált világidő** (*Universal Time Coordinated, UTC*).
  A Föld és a Nap mozgásából, azaz asztronómiai megfigyelésekből vezették le.
  - 1972-ben lépett a GMT (Greenwich Mean Time) helyébe azzal, hogy a másodperc a TAI szerint értendő.
  - A Föld mozgása enyhén szabálytalan, ezért alkalmanként beszúrnak egy szökő másodpercet.
  - 1958. január elsején a TAI és az UTC (egy megegyezés alapján) ugyanazt mutatta. Azóta az UTC mintegy 40 másodperces eltérést "szedett fel".
  - Az USA mérésügyi hivatalának (National Institute of Standards and Technology: NIST) rövidhullámú rádióadója (hívójele: WWV) folyamatosan ad frekvencia és időjelet 2.5, 5., 10, 15 és 20 MHz frekvencián. A jelek időbeni pontossága $\pm 1$msec, véletlen atmoszférikus ingadozások miatt $\pm 10$msec. (Geostacionárius műholdról $\pm 0.5$msec.)
- **Idő formátum**: legelterjedtebb: **Network Time Protocol** (NTP)
  Ez a formátum 8 bájtot használ, amelyből 4 az UTC másodperceket, 4 pedig a másodperc törtrészét tárolja, az utóbbit 232 psec felbontásban.
  1972 január elsején 00:00:00-kor 2 272 060 800.0 került a 8-bájtos számlálóba, ami az 1900. január elseje 00:00:00-tól eltelt másodpercek száma volt.
  Ez az ábrázolási mód 2036-ig „jó" (136 év a körülfordulási ciklusa).

Emlékeztető

**Példa:** Az óraszinkronizáció szükségessége/jelentősége: UNIX **make** program

Nagy programok forrásai fel vannak osztva részekre (pl 100 file).

Csak azokat kell újrafordítani, amelyekhez tartozó forrás megváltozott.

Ha a forrás időbélyege későbbi, például **input.c** (timestamp **2151**), és **input.o** (timestamp **2150**), akkor újra kell fordítani a forrás file-t.

De ha az editor és a compiler különböző gépen fut, akkor az időbélyegek értelmezésével baj lehet, ha az órák nincsenek szinkronban!

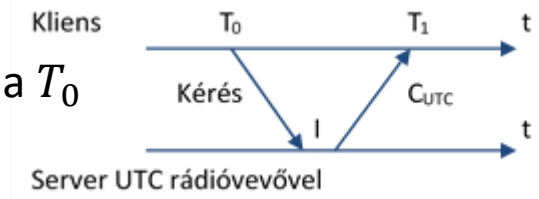
Ha azt tapasztaljuk, hogy a forrás időbélyege korábbi, mint a lefordítotté, azaz **output.c** (timestamp **2143**), és **output.o** (timestamp **2144**), akkor nem fordítunk.

De ha ennek az az oka, hogy az editort futtató gép órája késik két időegységet, akkor baj van!

# Órák szinkronizálása

**Berkeley algoritmus:** Aktív időszerver: rendszeresen lekérdezi a csomópontok óráját, átlagolja azokat, majd visszaküldi.

**Cristian algoritmus:** A szinkronizálást a kliens kezdeményezi a $T_0$ időpillanatban egy UTC rádióvevővel rendelkező szervernél.

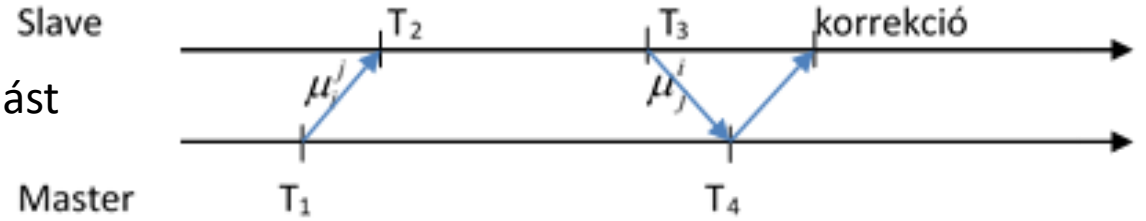



A kérés megérkezésekor, az interrupt kiszolgálását (*I*) követően a szerver lekérdezi az UTC rádiót, majd a lekérdezett $C_{UTC}$ értéket megküldjük a kliensnek. A $T_1$ időpillanatban megérkező óra adatot korrigálni kell az üzenettovábbítás idejével. Ha az üzenettovábbítás ideje mindkét irányban közel azonos, akkor a szükséges korrekció közelítése: $\sim \frac{T_1 - T_0 - I}{2}$.



**Megjegyzés:** Problémát okozhat, ha a $C_{UTC} + korrekció < T_1$, azaz a kliens óráját vissza kell állítani, mert siet. Ha kliens óra éppen egymást követő eseményekhez rendel időbélyeget, akkor előfordulhat, hogy visszaállítását követően későbbi eseményhez korábbi időbélyeget rendel, és ezzel az események időbeni **sorrendjét** látszólag megfordítja.
Ha ez a veszély fennáll, akkor az órát **nem szabad visszaállítani**, csak „**lassítani**" addig, amíg futása szinkronba nem kerül az $UTC$ rádió órájával.

**Master-slave algoritmusok:**




1. A master óra ($i$-jelű) szinkronizálást kezdeményez $T_1$ -ben.
Az órából kiolvasott érték hibája $e_1$. ($T_1 = C_i(T_1) + e_1$). A slave a $j$ csomópontban van. A $T_1$ -ben elküldött üzenet $\mu_i^j$ ideig utazik és $T_2$-ben érkezik meg. Ekkor $C_j(T_2)$ olvasható ki, amivel $T_2 = C_j(T_2) + e_2$ .
2. A slave kiszámolja a különbséget: $d_1 = C_j(T_2) - C_i(T_1)$

A vétel és az adás idejének összehasonlításával:

$$C_i(T_1) + e_1 + \mu_i^j = C_j(T_2) + e_2 \rightarrow d_1 = C_j(T_2) - C_i(T_1) = \mu_i^j + (e_1 - e_2)$$

ha a slave és a master órájának "átlagos" eltérését $\varepsilon_j$ jelöli, akkor $\varepsilon_j = (e_1 - e_2) + E_j^1$, ahol $E_j^1$ zajt modellez. (Az $(e_1 - e_2)$ pillanatértékek különbsége.) Ezzel

$$d_1 = C_j(T_2) - C_i(T_1) = \mu_i^j + (e_1 - e_2) = \mu_i^j + \varepsilon_j - E_j^1$$

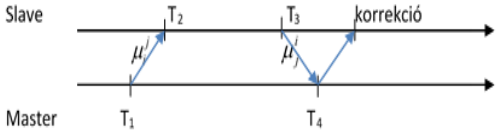
3. A slave elküldi órája állását a masternak $T_3 = C_j(T_3) + e_3$ időben.
Az üzenet $T_4 = C_i(T_4) + e_4$ időben érkezik $\mu_j^i$ utazási időt követően.
Ekkor a master kiszámítja a $d_2 = C_i(T_4) - C_j(T_3)$ különbséget.



A valóságos idők összevetésével:

$$C_j(T_3) + e_3 + \mu_j^i = C_i(T_4) + e_4 \rightarrow d_2 = C_i(T_4) - C_j(T_3) = \mu_j^i + (e_3 - e_4).$$

A slave és a master órájának $\varepsilon_j$ eltérését most $-\varepsilon_j = (e_3 - e_4) + E_j^2$ formában írhatjuk, ahol $E_j^2$ ugyancsak zajt modellez. Ezzel

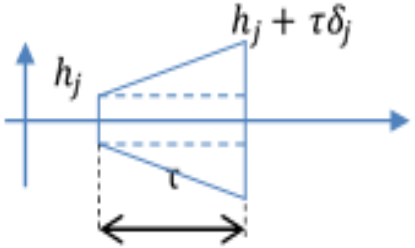



$$d_2 = \mu_j^i - \varepsilon_j - E_j^2.$$

A két különbség különbsége adja az órák "átlagos" eltérését, azaz a szükséges (slave) korrekció értékét:

$$(d_1 - d_2)/2 = \varepsilon_j + \left(\mu_i^j - \mu_j^i\right)/2 - \left(E_j^1 - E_j^2\right)/2 = \varepsilon_j + h_j,$$

A szinkronizálást követően megmaradó hiba a drift miatt változni/nőni fog. Ennek tartományát mutatja az alábbi ábra:




Az ábra alapján látható, hogy a korrekció után megmaradó hiba és a drift miatt $\tau$ idő multán, a legkedvezőtlenebb esetben az órák egymástól $2\left(\tau \max_j \delta_j + \max_j |h_j|\right)$ távolságra kerülhetnek.

**Megjegyzés:** (1) A közölt változatban a korrekció pontossága a **master** és a **slave** között a mérések többszöri megismétlésével és az eredmények átlagolásával javult.
(2) Amennyiben a master és a slave kommunikációjának további részletei ismertek (például LAN környezetben), akkor ennek felhasználásával a korrekció tovább finomítható.
(3) Ha $n$ processzor órájának frissítése a feladat, és minden slave $p$-szer lekérdezésre kerül, akkor a master-slave szinkronizáció kommunikáció igénye $(2p + 1)n$-nel jellemezhető $\tau$ időközönként.

**Példa: Tempo algoritmus**: master-slave szinkronizáció az elosztott Berkeley Unixban.

## Master oldal:

*Az alapalgoritmust N-szer lefuttatjuk:*
for *k*=1 to *N*
do
*Inicializálás:*
do

$$T_A \leftarrow C_i(now)$$
$$\forall j \neq i \text{ Send } T_A \text{ to } j$$

endo
*A slave-ektől kapott adatok feldolgozása:*
$\forall j \neq i$:
do

$$d_2^j \leftarrow C_i(now) - T_B$$
$$\Delta_j(k) \leftarrow \left(d_1^j - d_2^j\right)/2$$

endo
endo ; *rendelkezésre áll N differencia*
$\forall j - re$

$$\forall j \neq i:$$

do

$$\Delta_j = (1/N)\sum_{k=1}^{N} \Delta_j\,(k)$$
$$\text{Send}\left(\Delta_j\right) \text{ to j}$$

endo

## Slave oldal:

do

$$d_1^j \leftarrow C_j(now) - T_A$$
$$T_B \leftarrow C_j(now)$$
$$\text{Send } (T_B, d_1^j) \text{ to } i$$

endo

do

$$C_j(t) \leftarrow C_j(t) - \Delta_j$$

endo

## Elosztott óra szinkronizálási algoritmusok

Az elosztott óra-szinkronizáció esetében elsősorban az együttfutás biztosítása a cél.

Mindenki mindenkivel beszél, és az így szerzett információk alapján módosítják az órákat.

***I. Sorbarendezési módszer:*** Az üzenetek időbélyegzésén alapszik.

Az $i$ óra pontosságát korlátozza a ***drift*** mérték: $\forall t: \left|1 - \frac{d}{dt} C_i(t)\right| < \delta_i \ll 1$

A kommunikáció során a késleltetés $\mu < D < \eta$.

Minden óra a következő algoritmust implementálja:

- Minden lokális óra ütésére a lokális óra inkrementálódik;
- Minden órával rendelkező egység üzenetet küld a többinek legalább minden $\tau$ másodpercben. Minden üzenetnek része az időbélyeg $T_m$.
- Egy külső időbélyeg érkezését követően minden vevő az óráját $C_i(t) \leftarrow max(C_i(t), T_m + \mu)$ értékre állítja.

Az algoritmus a leggyorsabb, és nem szükségképpen a legpontosabb órához szinkronizál.

***II. Maximális hiba minimalizálása:***

Minden óra tudja, hogy helyes $[C_i(t_0) = t_0]$ egy adott intervallumban:

$[C_i(t) - E_i(t), C_i(t) + E_i(t)]$. $E_i(t)$ összetevői:

$\boldsymbol{\varepsilon_i}$alaphiba, vagy maradék-hiba a $\boldsymbol{\rho_i}$ reset időpontban.

$\boldsymbol{\mu_i^j}$ a késleltetés az $\boldsymbol{i}$ óra olvasása és a $\boldsymbol{j}$ óra frissítése között.

A $\boldsymbol{\delta_i}$ drift következtében a késleltetés okozta $\boldsymbol{\delta_i \mu_i^j}$hiba.

A kommunikáció időtartama és a drift okozta hibát egyaránt a $E_i(t)$ megnövelésével vesszük figyelembe: $(1 + \delta_i)\mu_i^j$.



*Maga az algoritmus:*

*Ha kérés érkezik $j \neq i$ -től*
do
$E_i(t) \leftarrow \varepsilon_i + (C_i(t) - \rho_i)\delta_i$
Send $(C_i(t), E_i(t))$ to $j$.
endo

Egyik szabály (az $i$-edik óra szemszögéből).

***Legalább egyszer $\tau$ időközönként***

$\forall j \neq i$: Request$\big(C_j(t), E_j(t)\big)$;
for $j \neq i$ do begin
Receive$\big(C_j(t), E_j(t)\big)$;
if $\big(C_j(t), E_j(t)\big)$ is consistent with $(C_i(t), E_i(t))$
then if $E_j(t) + (1 + \delta_i)\mu_j^i \leq E_i(t)$
then begin
$C_i(t) \leftarrow C_j(t)$
$\varepsilon_i \leftarrow E_j(t) + (1 + \delta_i)\mu_j^i$
$\rho_i \leftarrow C_j(t)$
end
else ignore it
end
endo

Másik szabály

A konzisztencia alatt azt értjük, hogy az időtengely mentén az intervallumok átlapolódóak. Ellenkező esetben azt feltételezzük, hogy valamelyik óra olyan mértékben hibás, figyelembe vétele rontaná a szinkronizálás eredményét.
A lekérdezett óra hiba-intervallum határát $E_j(t) + (1 + \delta_i)\mu_j^i$ formában használjuk az összehasonlítás folyamán, ill. a szinkronizálási maradékhiba betöltésekor, mert a mérés és a felhasználás között eltelt (kommunikációs) idő alatt a leolvasott $C_j(t)$ időbélyeg értékhez képest a *j*-edik óra hibája ennyivel nőtt.

### *III. Intervallumok metszése*

Ennél a módszernél összehasonlítjuk az órákat jellemző intervallumokat, és megkeressük a baloldali intervallumhatárok maximumát, és a jobboldali intervallumhatárok minimumát, és ha ez a két érték egy valóságos intervallum határait adja meg, akkor ez az intervallum lesz az órát jellemző új intervallum.

Az első szabály ugyanaz, mint előbb.

**Megjegyzés:** (1) Az intervallumok metszése módszer pontosabb, de kevésbé robusztus. (2) Ha $n$ processzor órájának frissítése a feladat, $2(n-1)n$ az elosztott óra szinkronizálás kommunikáció igénye $\tau$ időközönként.

### *Legalább egyszer τ időközönként*

$\forall j \neq i$: Request$\big(C_j(t), E_j(t)\big)$;
$\forall j \neq i$: Receive$\big(C_j(t), E_j(t)\big)$;
$\forall j \neq i\ L_j(t) \leftarrow \big(C_j(t) - E_j(t)\big)$; baloldali határ
$\forall j \neq i$: $R_j(t) \leftarrow \big(C_j(t) + E_j(t)\big) + (1 + \delta_i)\mu_j^i$
jobboldali határ
$\alpha \leftarrow max\big(L_j\big); \beta \leftarrow min\big(R_j\big)$
if $\alpha < \beta$
then
$\varepsilon_i \leftarrow \frac{1}{2}(\beta - \alpha)$;
$C_i(t) \leftarrow \frac{1}{2}(\alpha + \beta)$;
$\rho_i \leftarrow \frac{1}{2}(\alpha + \beta)$
end
else ignore them all
endo

## *A szinkronizációs üzenet jittere*

A *jitter*: $d_{max} - d_{min}$

- alkalmazói programból történő szinkronizáció esetén: $500\mu s$ …5ms
- operációs rendszer kernelből: $10\mu s$ …$100\mu s$
- kommunikációs vezérlő hardveréből: <$10\mu s$.

Bizonyítható, hogy $N$ óra esetén, ha $e$ értékű jittert (latency jittert) tételezünk fel a kommunikációban, akkor teljesen pontos órák esetén sem érhető el jobb együttfutás, mint

$$\Pi = e\left(1 - \frac{1}{N}\right)$$