Több mint negyven éves kutatómunkájának szakterülete a felületi kémia. A szilárd felületek szerkezetkutatásán belül azok reakciókészségét, a felületi folyamatok dinamikáját (az oszcilláló reakciók nem lineáris dinamikáját) vizsgálta. Tanulmányozta a heterogén katalízist, ezen belül a molekulák, atomok kemoszorbcióját. Kutatásainak eredményeit 692 szakdolgozatban, számos kézikönyvben publikálta munkatársaival együtt.

A szilárd-szilárd, szilárd-folyadék, szilárd-gáz, fázishatárokon történő kémiai változások jelentős szerepet játszanak számos vegyipari eljárásban (félvezetők gyártása, műtrágya-gyártás, különböző anyagok szintézise), üzemanyagcellák működésében, korrózió védelemben, az elektrokémiai folyamatok, meteorológiai folyamatok értelmezésében (az ózonpajzs vékonyodása okának tisztázásában). Kutatásainak eredményeit, kutatási módszereit az alapkutatásokban is és a vegyipari fejlesztésekben is sikerrel alkalmazzák.

Tudományos eredményeinek elismeréséül a világ minden táján számos egyetem, akadémiai intézet, tudományos társaság tagjának választotta, különböző díjakkal jutalmazták, melyek közül a legrangosabb a most elnyert kémiai Nobel-díj.

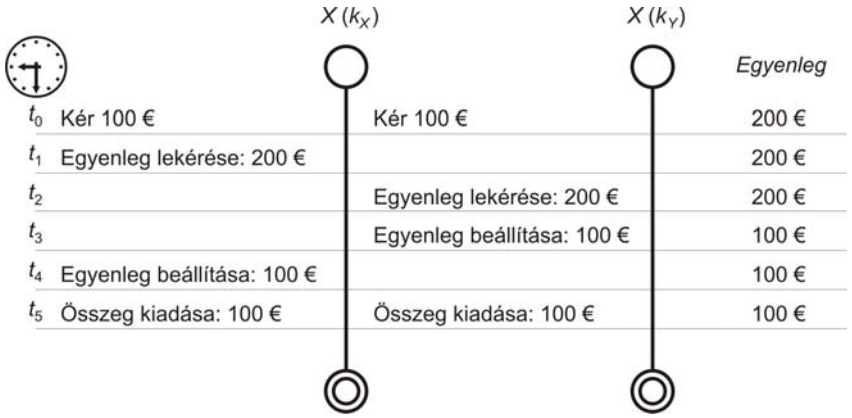
**Felhasznált forrásanyag**

1. A Nobel-díjasok kislexikona, Gondolat kiadó, Bp. 1974.
2. http://www.origo.hu/tudomany20071010

**M. E.**

# Folyamatszálak szinkronizálása

Miért kell szinkronizálni? Kérdésünkre egyszerű választ ad a következő példa: Két személy, *X* és *Y* ugyanarra a bankszámlára kiállított bankkártyával rendelkezik ($k_X$, $k_Y$). Mindketten egyszerre lépnek oda egy-egy bankautomatához ($t_0$ időpont) és fel szeretnének venni egyenként 100 €-t. A bankszámlán 200 € van. A 1. ábrán bemutatjuk, hogy mi történik szinkronizálás nélkül.




1. ábra
*„Bankrablás" – szinkronizálás hiánya*

Szinkronizálás hiányában a két folyamatszál lekéri az egyenleget a $t_1$, illetve a $t_2$ időpontban (200 €), egyenként levonja a kért 100 €-t, a $t_3$, illetve a $t_4$ időpontban beállítja az új egyenleget (200 € - 100 € = 100 €), majd a $t_5$ időpontban kiadja mindkét személynek a 100 €-t. A folyamat eredménye az lett, hogy az eredetileg 200 €-t tartalmazó bankszámláról mindkét személynek kiad 100-100 €-t, és a bankszámlán is megmarad 100 €. (Ez egy lehetséges forgatókönyv, a kommunikáció sebessége a bank és a bankautomata közötti összeköttetés sebességétől függ, a műveletek sorrendje meghatározott, de a két szálon egymáshoz viszonyítva semmi sem biztosítja az egymásutániságot vagy precedenciát, a két szál között semmiféle kommunikáció nincs. Az ilyen problémákat *race condition*-nak, *versengési feltétel*nek nevezzük: két vagy több szál egyszerre ugyanazért az erőforrásért verseng.) Na jó, nekünk jól jön a szinkronizálás hiánya, de a banknak...? – kész csőd.

A szálak együttműködésének megértéséhez szükséges megértenünk az *atomiság* (*elemiség*) problémáját. Egy művelet vagy művelet sorozat atomi (elemi), ha tovább nem bontható más tevékenységekre. Ha egy szál elemi utasítást hajt végre, akkor az azt jelenti, hogy a többi szál nem kezdte el annak végrehajtását vagy már befejezte azt. Nem állhat fenn olyan eset, hogy egy szál egy másikat „egy adott utasítás végrehajtásán érjen". Ha semmilyen szinkronizálást nem alkalmazunk a szálak közt, akkor szinte egy műveletet sem mondhatunk atominak.

Lássunk egy másik egyszerű példát is az atomiságra: az *A* globális változó értékét növeljük meg két különböző szálban (`inc(A)`, `inc(A)` utasítást kell kiadni minden egyes szálban, az elvárt eredmény az, hogy az *A* értéke 2-vel nőjön). Ez az egyszerű utasítás assembly szinten három különböző utasításra bomlik le:

1. Olvasd *A*-t a memóriából egy processzor-regiszterbe.
2. Adj hozzá 1-et a processzor-regiszter értékéhez.
3. Írd a regiszter értékét vissza a memóriába.

Ha csak egyetlen processzorunk van, akkor egyszerre csak egy szál dolgozik, de az operációs rendszer ütemezője másodpercenként kb. 18-szor vált köztük. Az ütemező bármelyik pillanatban leállíthatja egyik szál futását és másikat indíthat el (az ütemezés *preemptív*). Az operációs rendszer nem vár a szál engedélyére, hogy mikor függeszthető fel azt és indíthat el helyette egy másikat. Így a váltás bármely két processzor-utasítás között megtörténhet.

Képzeljük el, hogy két szál ugyanazt a kódrészletet (az *A* változó értékének növelését) hajtja végre egy egy processzoros gépen (a szálak legyenek *X* és *Y*). Szerencsés esetben, ha a program jól működik, az ütemezési műveletek elkerülhetik a kritikus szakaszt és a várt eredményt kapjuk: *A* értékét megnöveltük 2-vel (legyen *A* eredeti értéke 1).

| *Az X szál által végrehajtott utasítások* | *Az Y szál által végrehajtott utasítások* |
|---|---|
| `<Egyéb utasítások>` | `Szál felfüggesztve` |
| `Olvasd A értékét a memóriából egy processzor-regiszterbe (1)` | `Szál felfüggesztve` |
| `Adj hozzá 1-et a regiszter értékéhez` | `Szál felfüggesztve` |
| `Írd a regiszter értékét vissza a memóriába (2)` | `Szál felfüggesztve` |
| `<Egyéb utasítások>` | `Szál felfüggesztve` |
| **`Szálváltás`** | **`Szálváltás`** |

| | |
|---|---|
| Szál felfüggesztve | <Egyéb utasítások> |
| Szál felfüggesztve | Olvasd *A* értékét a memóriából egy processzor-regiszterbe |
| Szál felfüggesztve | Adj hozzá 1-et a regiszter értékéhez |
| Szál felfüggesztve | Írd a regiszter értékét vissza a memóriába (3) |
| Szál felfüggesztve | <Egyéb utasítások> |

Viszont ez nem jelenti azt, hogy a várt eredményt garantáltan megkapjuk (Murphy törvényét ismerve biztos, hogy az *A* értéke csak 1-el fog megnőni).

| *Az X szál által végrehajtott utasítások* | *Az Y szál által végrehajtott utasítások* |
|---|---|
| <Egyéb utasítások> | Szál felfüggesztve |
| Olvasd *A* értékét a memóriából egy processzor-regiszterbe (1) | Szál felfüggesztve |
| Adj hozzá 1-et a regiszter értékéhez (2) | Szál felfüggesztve |
| **Szálváltás** | **Szálváltás** |
| Szál felfüggesztve | <Egyéb utasítások> |
| Szál felfüggesztve | Olvasd *A* értékét a memóriából egy processzor-regiszterbe (1) |
| Szál felfüggesztve | Adj hozzá 1-et a regiszter értékéhez (2) |
| Szál felfüggesztve | Írd a regiszter értékét vissza a memóriába (2) |
| **Szálváltás** | **Szálváltás** |
| Írd a regiszter értékét vissza a memóriába (2) | Szál felfüggesztve |
| <Egyéb utasítások> | Szál felfüggesztve |

Láthatjuk tehát, hogy szinkronizálás vagy konkurencia szabályozás (*concurrency control*) nélkül súlyos problémák léphetnek fel elsősorban a következő műveletek esetében:

- megosztott erőforráshoz való hozzáféréskor
- nem VCL (grafikus felület) szálból a VCL nem szálbiztonságos részeinek elérése (szinkronizálás a grafikus felülettel, ablakkal)
- különálló szálból grafikus művelet végrehajtásánál

A VCL sem rendelkezik védelemmel a szinkronizációs-konfliktusok ellen. Ez azt jelenti, hogy a szálváltás akkor is megtörténhet, amikor egy vagy több szál VCL kódot hajt végre. Egy szálváltás a nem megfelelő pillanatban megsértheti nem csak az adatot, hanem a kapcsolatokat is a komponensek között.

Például a VCL-szál egy dialógusablakot nyit meg vagy egy lemezre írást kezd el, és felfüggesztődik, közben egy másik szál megváltoztat valamilyen megosztott adatot. A fő

VCL-szál továbbindításkor észreveszi, hogy az adat megváltozott a dialógusablak felbukkanásának vagy a lemezre írás eredményeként. Rosszul értelmezi a helyzetet, rossz pontról folytatja a műveleteket.

A közös erőforrások használata, valamint a szálak közötti közös memória használatának biztonsága és helyessége érdekében a folyamatok aszinkron, teljesen szabad futását korlátozni kell. Ezeket a korlátozásokat nevezzük *szinkronizálás*nak.

Szinkronizálást megvalósíthatunk a következő szinkronizációs primitívekkel:

- Csatorna
- Mutex
- Feltételes változó
- Kritikus szakasz (kritikus zóna)
- Szemafor
- Monitor

**Csatorna**

A folyamatok közötti kommunikáció úgynevezett randevúk segítségével is megvalósítható. A randevúk lényege, hogy az a folyamat, amely elsőnek érkezik, mindaddig fel lesz függesztve (vár), míg a társa oda nem ér. Amikor a randevú teljes, akkor lefut mind a két folyamat. A folyamatok közötti kommunikáció csatornák segítségével valósulhat meg.

A csatornákat Hoare vezette be 1985-ben. A lényegük az, hogy üzenetet küldhetünk, vagy üzenet fogadhatunk egy csatornán:

- `ch!e` – az *e* értéket elküldjük a *ch* csatornán
- `ch?v` – a *v* értéket vesszük a *ch* csatornáról

**Mutex**

A mutex (*mutual exclusion*) változó tulajdonképpen egy bináris szemafor, két lehetséges állapota van:

- lezárt (0) – egy szál tulajdona soha sem lehet egyszerre több szálnak a tulajdona. Ha egy szál egy lezárt mutexet szeretne megkapni, akkor várnia kell, amíg azt a foglaló szál felszabadítja.
- nyitott (1)– egyetlen szálnak sem a tulajdona

Egy mutex változón a következő műveleteket végezhetjük:

- Inicializálás (statikus vagy dinamikus)
- lezárás (hozzáférés igénylése a védett erőforráshoz = mutex igénylése)
- nyitás (felszabadítja az erőforrást)
- a mutex változónak megsemmisítése

**Feltételes változó**

A feltételes változók szinkronizációs és kommunikációs objektumok egy feltétel teljesülésére várakozó szál és egy feltételt teljesítő szál között.

A feltételes változóhoz hozzá van rendelve egy:

- predikátum – a feltétel aminek teljesülnie kell
- mutex változó – a mutex változó biztosítja, hogy a feltétel ellenőrzése és a várakozás, vagy a feltétel ellenőrzése és teljesülésének jelzése atomi műveletként fusson.

Egy feltételes változón a következő műveleteket végezhetjük:

- Inicializálás (statikus vagy dinamikus)
- Várakozás (WAIT) – a szál várakozik amíg kívülről jelzik a feltétel teljesülését
- Jelzés (NOTYFY) – az aktuális szál jelez az összes szálnak, amelyek várják a feltétel beteljesedését
- Megsemmisítés

**Kritikus szakasz**

Ha egy szál belép egy kritikus szakaszba, akkor lefoglal egy erőforrást, és más szál nem férhet hozzá az adott erőforráshoz, ameddig a szál ki nem lép a kritikus szakaszból.

Egy jól definiált kritikus szakasz a következő tulajdonságokkal rendelkezik:

- Egy adott időpontban egyetlen szál található a kritikus szakaszban, bármely más szál, amely hozzá akar férni a kritikus erőforráshoz, várakozik, míg az eredeti szál ki nem lép a kritikus szakaszból.
- A szálak relatív sebességei nem ismertek.
- Bármely szál leállítása csak a kritikus szakaszon kívül történhet.
- Egyetlen szál sem fog végtelen időt várni a kritikus szakaszban.

A kritikus szakasz legegyszerűbb megvalósítása egy bináris szemafor segítségével történik.

**Szemafor**

A folyamatok szinkronizálására speciális nyelvi elemeket kell bevezetni. A legelőször bevezetett nyelvi elem a szemafor volt, amelyet Dijkstra mutatott be 1968-ban a kölcsönös kizárás problémájának megoldására. Ez az eszköz a nevét a vasúti jelzőberendezésről kapta, logikai hasonlósága miatt.

A szemafor általánosan egy pozitív egész értéket vehet fel. Speciális a bináris szemafor, amelynek értéke csak 0 és 1 lehet. A lehetséges műveletek neve `wait` és `signal`. A műveletek hatása a következő:

- *wait(s)*: ha $s > 0$, akkor $s := s - 1$, különben blokkolja ezt a folyamatot $s$-en;
- *signal(s)*: ha van blokkolt folyamat $s$-en, akkor indít közülük egyet, különben $s := s + 1$.

Lényeges, hogy ezek a műveletek oszthatatlanok, azaz végrehajtásuk közben nem történhet folyamatváltás. Ugyancsak lényeges a szemafor inicializálása is, mert itt határozzuk meg, hogy egyszerre hányan férhetnek hozzá az erőforráshoz.

**Monitor**

A monitorokat Hoare vezette be 1974-ben. A monitor objektum egy több szál által használt eljárás nem párhuzamos végrehajtását teszi lehetővé. A monitor tulajdonképpen ötvözi az objektumorientált programozást a szinkronizációs metódusokkal.

Egy monitor objektum részei:

- osztott adat
- ezeket az adatokat feldolgozó eljárások
- monitort inicializáló metódusok

Mindegyik eljáráshalmazt egy monitor kontrolál. A többszálas alkalmazás futásakor a monitor egyetlen szálnak engedélyezi egy adott időpontban az eljárás végrehajtását. Ha egy szál éppen egy monitor által kontrolált eljárást akar futtatni, akkor az lefoglalja a monitort. Ha a monitor már foglalt, akkor várakozik, amíg a monitort lefoglaló szál befejezi a adott eljárás végrehajtását és felszabadítja a monitort.

**Kovács Lehel**

# A sötét anyag és a sötét energia „megvilágítása”

I. rész

A 70-es évek végén sikerült feltérképezni a szép, szabályos spirál galaxisokat. Szinte szemmel látható volt, hogy az egyes csillagok a galaxis középpontja körül keringenek. Kiszemelünk egy csillagot. Megmérjük a középponttól mért r távolságát, és megbecsüljük azon csillagok együttes M(r), tömegét, amelyek ezen r távolságon belül láthatók. Az egyenletes körmozgásra vonatkozó

$$m\,v^2\,/r = G\,m\,M(r)\,/r^2$$

alakú Newton-egyenletből ki lehet számítani a csillag v keringési sebességét.

Itt G a Newton-féle gravitációs állandó, m pedig a csillag tömege, ami azonban kiesik az egyenletből:

$$v^2 = G\,M(r)\,/r$$

Innen a v sebességet kiszámították.

A csillag fényének színképében felismerhetők a hidrogén színképvonalai. Ezek azonban a laboratóriumban megfigyelhető vonalakhoz képest eltolódva jelentkeztek. Ebből, az ún. Doppler eltolódásból ki lehet számítani a csillag keringési sebességét.

A v sebességet így is kiszámították.

**A két különböző módon meghatározott sebesség azonban nem egyezett meg!**

Mi lehet az oka a különbségnek?

Kiderült, hogy a két sebességérték „egyenlővé tehető”, ha feltételezzük, hogy a galaxisban jelen van valamilyen nem látható „sötét anyag” is. Ekkor a fenti képletben M(r) helyébe az $(M(r)+M_{DM}(r))$ összeget kell írni, ahol $M_{DM}(r)$ a feltételezett sötét anyag (Dark Matter) azon részének tömege, ami az r sugáron belül helyezkedik el. Az elmúlt évek során igen sok galaxis esetén végeztek el hasonló elemzést. Az eredmény az lett, hogy a gala-