Megelőzési gráfok és teszt a konfliktus-sorbarendezhetőségre

- **Alapötlet: ha valahol konfliktusban álló műveletek szerepelnek S-ben, akkor az ezeket a műveleteket végrehajtó tranzakcióknak ugyanabban a sorrendben kell előfordulniuk a konfliktus-ekvivalens soros ütemezésekben, mint ahogyan az S-ben voltak.**

- **Tehát a konfliktusban álló műveletpárok megszorítást adnak a feltételezett konfliktusekvivalens soros ütemezésben a tranzakciók sorrendjére.**

- **Ha ezek a megszorítások nem mondanak ellent egymásnak, akkor találhatunk konfliktusekvivalens soros ütemezést. Ha pedig ellentmondanak egymásnak, akkor tudjuk, hogy nincs ilyen soros ütemezés.**

Megelőzési gráfok és teszt a konfliktus-sorbarendezhetőségre

- **Adott a T_1 és T_2, esetleg további tranzakcióknak egy S ütemezése. Azt mondjuk, hogy T_1 *megelőzi* T_2-t, ha van a T_1-ben olyan A_1 művelet és a T_2-ben olyan A_2 művelet, hogy**

1. **A_1 megelőzi A_2-t S-ben,**
2. **A_1 és A_2 ugyanarra az adatbáziselemre vonatkoznak, és**
3. **A_1 és A_2 közül legalább az egyik írás művelet.**

- **Másképpen fogalmazva: A_1 és A_2 konfliktuspárt alkotna, ha szomszédos műveletek lennének. Jelölése: $T_1 <_S T_2$.**

- **Látható, hogy ezek pontosan azok a feltételek, amikor nem lehet felcserélni A_1 és A_2 sorrendjét. Tehát A_1 az A_2 előtt szerepel bármely S-sel konfliktusekvivalens ütemezésben. Ebből az következik, hogy ha ezek közül az ütemezések közül az egyik soros ütemezés, akkor abban T_1-nek meg kell előznie T_2-t.**

- **Ezeket a megelőzéseket a *megelőzési gráfban* (precedence graph) összegezhetjük. A megelőzési gráf csúcsai az S ütemezés tranzakciói. Ha a tranzakciókat T_i-vel jelöljük, akkor a T_i-nek megfelelő csúcsot az i egész jelöli. Az i csúcsból a j csúcsba akkor vezet irányított él, ha $T_i <_S T_j$.**

Lemma

S_1, S_2 konfliktusekvivalens $\Rightarrow$ gráf(S_1)=gráf(S_2)

Bizonyítás:

Tegyük fel, hogy gráf(S_1) $\neq$ gráf(S_2)

$\Rightarrow \exists$ olyan $T_i \rightarrow T_j$ él, amely gráf(S_1)-ben benne van, de gráf(S_2)-ben nincs benne, (vagy fordítva.)

$\Rightarrow S_1 = \ldots p_i(A) \ldots q_j(A) \ldots$ p_i, q_j

$S_2 = \ldots q_j(A) \ldots p_i(A) \ldots$ **konfliktusos pár**

$\Rightarrow S_1, S_2$ nem konfliktusekvivalens. Q.E.D.

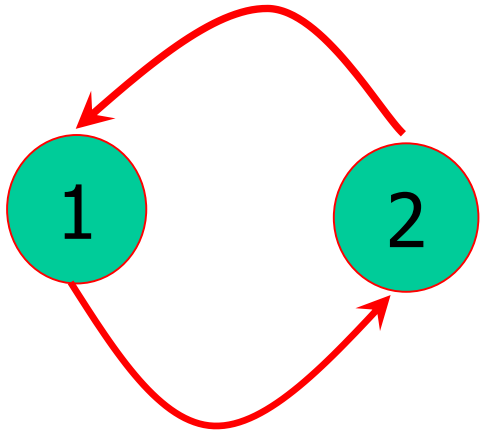
Megjegyzés:

gráf(S_1)=gráf(S_2) $\not\Rightarrow$ S_1, S_2 konfliktusekvivalens

Ellenpélda:

$S_1 = w_1(A)\ r_2(A)\ w_2(B)\ r_1(B)$

$S_2 = r_2(A)\ w_1(A)\ r_1(B)\ w_2(B)$



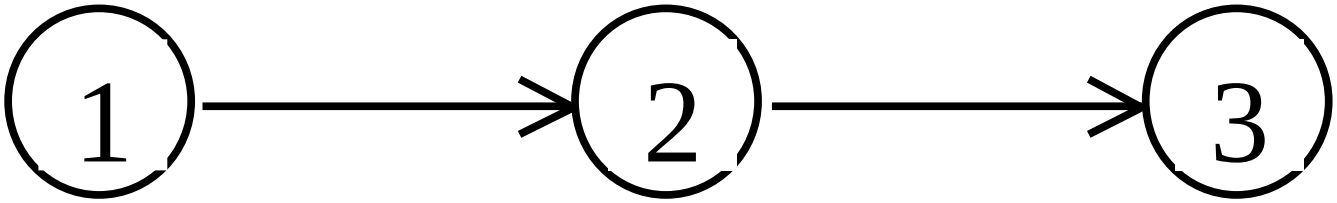
Nem lehet semmit sem cserélni!

Megelőzési gráfok és teszt a konfliktus-sorbarendezhetőségre

- ***Példa.*** **A következő S ütemezés a T_1, T_2 és T_3 tranzakciókat tartalmazza:**

S: $r_2(A); r_1(B); w_2(A); r_3(A); w_1(B); w_3(A); r_2(B); w_2(B);$

- **Az *S* ütemezéshez tartozó megelőzési gráf a következő:**



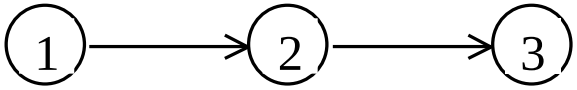
TESZT: Ha az S megelőzési gráf tartalmaz irányított kört, akkor S nem konfliktus-sorbarendezhető, ha nem tartalmaz irányított kört, akkor S konfliktus-sorbarendezhető, és a csúcsok bármelyik topologikus sorrendje megadja a konfliktusekvivalens soros sorrendet.

Megelőzési gráfok és teszt a konfliktus-sorbarendezhetőségre

- **Egy körmentes gráf csúcsainak *topologikus sorrendje* a csúcsok bármely olyan rendezése, amelyben minden a → b élre az a csúcs megelőzi a b csúcsot a topologikus rendezésben.**

S: $r_2(A); r_1(B); w_2(A); r_3(A); w_1(B); w_3(A); r_2(B); w_2(B);$

- **Az S ütemezéshez tartozó megelőzési gráf topologikus sorrendje: (T1, T2, T3).**



S′: $r_1(B); w_1(B); r_2(A); w_2(A); r_2(B); w_2(B); r_3(A); w_3(A);$

- Hogy lehet S-ből S'-t megkapni szomszédos elemek cseréjével?

1. $r_1(B); r_2(A); w_2(A); w_1(B); r_3(A); r_2(B); w_3(A); w_2(B);$
2. $r_1(B); r_2(A); w_1(B); w_2(A); r_2(B); r_3(A); w_2(B); w_3(A);$
3. $r_1(B); w_1(B); r_2(A); w_2(A); r_2(B); w_2(B); r_3(A); w_3(A);$

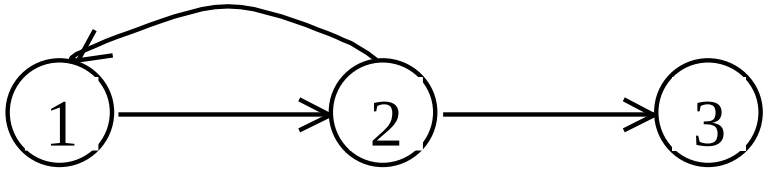
Megelőzési gráfok és teszt a konfliktus-sorbarendezhetőségre

- ***Példa.*** **Tekintsük az alábbi két ütemezést:**

S: $r_2(A); r_1(B); w_2(A); r_3(A); w_1(B); w_3(A); r_2(B); w_2(B);$

S_1: $r_2(A); r_1(B); w_2(A); r_2(B); r_3(A); w_1(B); w_3(A); w_2(B);$

- **Az** $r_2(A) \ldots w_3(A)$ **miatt** $T_2 <_{S1} T_3$.
- **Az** $r_1(B) \ldots w_2(B)$ **miatt** $T_1 <_{S1} T_2$.
- **Az** $r_2(B) \ldots w_1(B)$ **miatt** $T_2 <_{S1} T_1$.
- **Az** S_1 **ütemezéshez tartozó megelőzési gráf a következő:**



- **Ez a gráf nyilvánvalóan tartalmaz kört, ezért** S_1 **nem konfliktus-sorbarendezhető, ugyanis láthatjuk, hogy bármely konfliktusekvivalens soros ütemezésben** T_1**-nek** T_2 **előtt is és után is kellene állnia, tehát nem létezik ilyen ütemezés.**

Miért működik a megelőzési gráfon alapuló tesztelés?

Ha van kör a gráfban, akkor cserékkel nem lehet soros ütemezésig eljutni.

Ha létezik a $T_1 \rightarrow T_2 \rightarrow \ldots \rightarrow T_n \rightarrow T_1$ n darab tranzakcióból álló kör, akkor a feltételezett soros sorrendben T_1 műveleteinek meg kell előzniük a T_2-ben szereplő műveleteket, amelyeknek meg kell előzniük a T_3-belieket és így tovább egészen T_n-ig. De T_n műveletei emiatt a T_1-beliek mögött vannak, ugyanakkor meg is kellene előzniük a T_1-belieket a $T_n \rightarrow T_1$ él miatt. Ebből következik, hogy ha a megelőzési gráf tartalmaz kört, akkor az ütemezés nem konfliktus-sorbarendezhető.

Miért működik a megelőzési gráfon alapuló tesztelés?

Ha nincs kör a gráfban, akkor cserékkel el lehet jutni egy soros ütemezésig.

A bizonyítás az ütemezésben részt vevő tranzakciók száma szerinti indukcióval történik:

Alapeset: Ha n = 1, vagyis csak egyetlen tranzakcióból áll az ütemezés, akkor az már önmagában soros, tehát konfliktus-sorbarendezhető.

Indukció: Legyen S a $T_1, T_2, \ldots, T_n$ n darab tranzakció műveleteiből álló ütemezés, és S-nek körmentes megelőzési gráfja van.

Ha egy véges gráf körmentes, akkor van egy olyan csúcsa, amelybe nem vezet él. Legyen a T_i egy ilyen csúcs.

Mivel az i csomópontba nem vezet él, ezért nincs S-ben olyan A művelet, amely

1. **valamelyik T_j ($i \neq j$) tranzakcióra vonatkozik,**
2. **T_i valamely műveletét megelőzi, és**
3. **ezzel a művelettel konfliktusban van.**

Így T_i minden műveletét S legelejére mozgathatjuk át, miközben megtartjuk a sorrendjüket:

S1: (T_i műveletei) (a többi $n-1$ tranzakció műveletei)

Miért működik a megelőzési gráfon alapuló tesztelés?

Ha nincs kör a gráfban, akkor cserékkel el lehet jutni egy soros ütemezésig.
(Folytatás)

S1: (T_i műveletei) (a többi $n-1$ tranzakció műveletei)

Most tekintsük S1 második részét. Mivel ezek a műveletek egymáshoz viszonyítva ugyanabban a sorrendben vannak, mint ahogyan S-ben voltak, ennek a második résznek a megelőzési gráfját megkapjuk S megelőzési gráfjából, ha elhagyjuk belőle az i csúcsot és az ebből a csúcsból kimenő éleket.

Mivel az eredeti körmentes volt, az elhagyás után is az marad, azaz a második rész megelőzési gráfja is körmentes.

Továbbá, mivel a második része n-1 tranzakciót tartalmaz, alkalmazzuk rá az indukciós feltevést.

Így tudjuk, hogy a második rész műveletei szomszédos műveletek szabályos cseréivel átrendezhetők soros ütemezéssé. Ily módon magát S-et alakítottuk át olyan soros ütemezéssé, amelyben T_i műveletei állnak legelöl, és a többi tranzakció műveletei ezután következnek valamilyen soros sorrendben. Q.E.D.

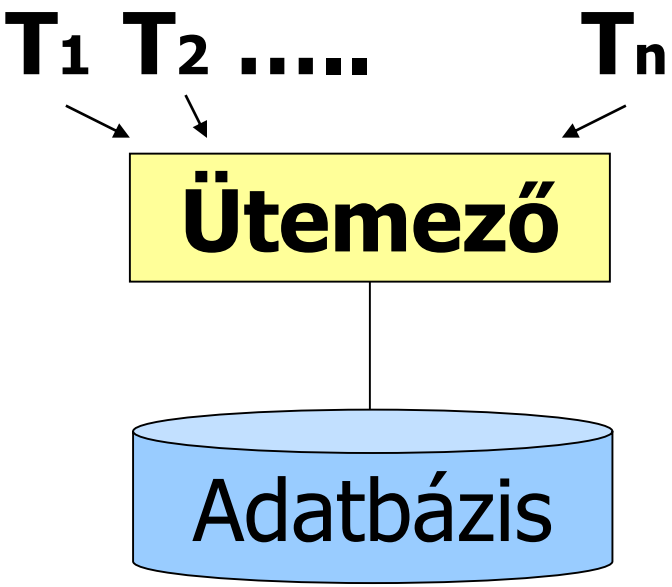
Az ütemező eszközei a sorbarendezhetőség elérésére

Passzív módszer:

- **hagyjuk a rendszert működni,**
- **az ütemezésnek megfelelő gráfot tároljuk,**
- **egy idő után megnézzük, hogy van-e benne kör,**
- **és ha nincs, akkor szerencsénk volt, jó volt az ütemezés.**

Az ütemező eszközei a sorbarendezhetőség elérésére

Aktív módszer: **az ütemező beavatkozik, és megakadályozza, hogy kör alakuljon ki.**



Az ütemező eszközei a sorbarendezhetőség elérésére

- Az ütemezőnek több lehetősége is van arra, hogy kikényszerítse a sorbarendezhető ütemezéseket:

1. zárak (ezen belül is még: protokoll elemek, pl. 2PL)
2. időbélyegek (time stamp)
3. érvényesítés

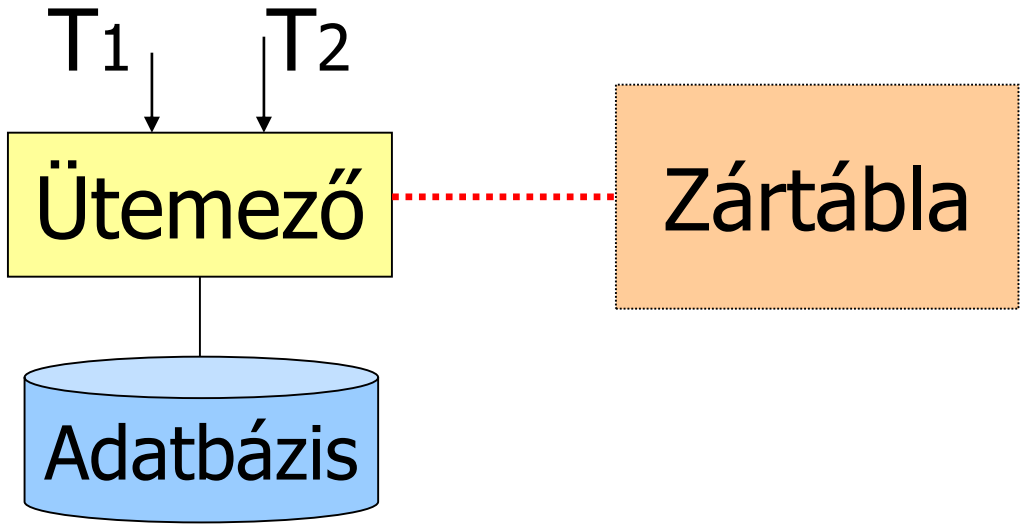
- Fő elv: inkább legyen szigorúbb és ne hagyjon lefutni egy olyan ütemezést, ami sorbarendezhető, mint hogy fusson egy olyan, ami nem az.

A sorbarendezhetőség biztosítása zárakkal

Két új műveletet vezetünk be:

- l_i (A) : kizárólagos zárolás (exclusive lock)
- u_i (A): a zár elengedése (unlock)

A tranzakciók zárolják azokat az adatbáziselemeket, amelyekhez hozzáférnek, hogy megakadályozzák azt, hogy ugyanakkor más tranzakciók is hozzáférjenek ezekhez az elemekhez, mivel ekkor felmerülne a nem sorbarendezhetőség kockázata.



A zárak használata

- **A *zárolási ütemező* a konfliktus-sorbarendezhetőséget követeli meg, (ez erősebb követelmény, mint a sorbarendezhetőség).**

***Tranzakciók konzisztenciája* (consistency of transactions):**

1. **A tranzakció csak akkor olvashat vagy írhat egy elemet, ha már korábban zárolta azt, és még nem oldotta fel a zárat.**
2. **Ha egy tranzakció zárol egy elemet, akkor később azt fel kell szabadítania.**

***Az ütemezések jogszerűsége* (legality of schedules):**

1. **nem zárolhatja két tranzakció ugyanazt az elemet, csak úgy, ha az egyik előbb már feloldotta a zárat.**

A zárak használata

- **Kibővítjük a jelöléseinket a zárolás és a feloldás műveletekkel:**
- **$l_i(X)$: a T_i tranzakció az X adatbáziselemre *zárolást kér* (lock).**
- **$u_i(X)$: a T_i tranzakció az X adatbáziselem *zárolását feloldja* (unlock).**
- **Konzisztencia:**
- **Ha egy T_i tranzakcióban van egy $r_i(X)$ vagy egy $w_i(X)$ művelet, akkor van korábban egy $l_i(X)$ művelet, és van később egy $u_i(X)$ művelet, de a zárolás és az írás/olvasás között nincs $u_i(X)$.**

Ti: … $l_i(X)$ … $r/w_i(X)$ … $u_i(X)$ …

A zárak használata

- **Jogszerűség:**
- **Ha egy ütemezésben van olyan $l_i(X)$ művelet, amelyet $l_j(X)$ követ, akkor e két művelet között lennie kell egy $u_i(X)$ műveletnek.**

$$S = \ldots\ldots\ldots l_i(X) \ldots\ldots\ldots\ldots u_i(X) \ldots\ldots\ldots$$

$\longleftrightarrow$

nincs $l_j(X)$

A zárak használata

T_1: $l_1(A); r_1(A); A := A+100; w_1(A); u_1(A);$
$l_1(B); r_1(B); B := B+100; w_1(B); u_1(B);$

T_2: $l_2(A); r_2(A); A := A*2; w_2(A); u_2(A);$
$l_2(B); r_2(B); B := B*2; w_2(B); u_2(B);$

Mindkét tranzakció konzisztens.

A zárak használata

T_1	T_2	A	B
$l_1(A)$; $r_1(A)$;		25	
A := A+100;			
$w_1(A)$; $u_1(A)$;		125	
	$l_2(A)$; $r_2(A)$;	125	
	A := A*2;		
	$w_2(A)$; $u_2(A)$;	250	
	$l_2(B)$; $r_2(B)$;		25
	B := B*2;		
	$w_2(B)$; $u_2(B)$;		50
$l_1(B)$; $r_1(B)$;			50
B := B+100;			
$w_1(B)$; $u_1(B)$;			150

Ez az ütemezés jogszerű,

de nem sorba rendezhető.

A zárolási ütemező

- **A zároláson alapuló ütemező feladata, hogy akkor és csak akkor engedélyezze a kérések végrehajtását, ha azok jogszerű ütemezéseket eredményeznek.**

- **Ezt a döntést segíti a zártábla, amely minden adatbáziselemhez megadja azt a tranzakciót, ha van ilyen, amelyik pillanatnyilag zárolja az adott elemet.**

- **A zártábla szerkezete (egyféle zárolás esetén): Zárolások(elem, tranzakció) relációt, ahol a T tranzakció zárolja az X adatbáziselemet.**

A zárolási ütemező

T_1: $l_1(A)$; $r_1(A)$; A := A+100; $w_1(A)$; $l_1(B)$;

$u_1(A)$; $r_1(B)$; B := B+100; $w_1(B)$; $u_1(B)$;

T_2: $l_2(A)$; $r_2(A)$; A := A*2; $w_2(A)$; $l_2(B)$;

$u_2(A)$; $r_2(B)$; B := B*2; $w_2(B)$; $u_2(B)$;

T_1	T_2	A	B
$l_1(A)$; $r_1(A)$;		25	
A := A+100;			
$w_1(A)$; $l_1(B)$; $u_1(A)$;		125	
	$l_2(A)$; $r_2(A)$;	125	
	A := A*2;		
	$w_2(A)$;	250	
	$l_2(B)$; elutasítva		
$r_1(B)$; B := B+100;			25
$w_1(B)$; $u_1(B)$;			125
	$l_2(B)$; $u_2(A)$; $r_2(B)$;		125
	B := B*2;		
	$w_2(B)$; $u_2(B)$;		250

Mivel T_2-nek várakoznia kellett, ezért B-t akkor szorozza meg 2-vel, miután T_1 már hozzáadott 100-at, és ez konzisztens adatbázis-állapotot eredményez.

A kétfázisú zárolás (two-phase locking, 2PL)

T_i = ... $l_i(A)$ ..... $u_i(A)$ .....

nincs unlock

nincs lock

Minden tranzakcióban minden zárolási művelet megelőzi az összes zárfeloldási műveletet.

A kétfázisú zárolás (two-phase locking, 2PL)

T_1	T_2	A	B
$l_1(A)$; $r_1(A)$;		25	
A := A+100;			
$w_1(A)$; $u_1(A)$;		125	
	$l_2(A)$; $r_2(A)$;	125	
	A := A*2;		
	$w_2(A)$; $u_2(A)$;	250	
	$l_2(B)$; $r_2(B)$;		25
	B := B*2;		
	$w_2(B)$; $u_2(B)$;		50
$l_1(B)$; $r_1(B)$;			50
B := B+100;			
$w_1(B)$; $u_1(B)$;			150

Ez az ütemezés jogszerű,

de nem sorba rendezhető.

A tranzakciók nem kétfázisúak!

A kétfázisú zárolás (two-phase locking, 2PL)

T_1: $l_1(A)$; $r_1(A)$; A := A+100; $w_1(A)$; $l_1(B)$;

$u_1(A)$; $r_1(B)$; B := B+100; $w_1(B)$; $u_1(B)$;

T_2: $l_2(A)$; $r_2(A)$; A := A*2; $w_2(A)$; $l_2(B)$;

$u_2(A)$; $r_2(B)$; B := B*2; $w_2(B)$; $u_2(B)$;

T_1	T_2	A	B
$l_1(A)$; $r_1(A)$;		25	
A := A+100;			
$w_1(A)$; $l_1(B)$; $u_1(A)$;		125	
	$l_2(A)$; $r_2(A)$;	125	
	A := A*2;		
	$w_2(A)$;	250	
	$l_2(B)$; elutasítva		
$r_1(B)$; B := B+100;			25
$w_1(B)$; $u_1(B)$;			125
	$l_2(B)$; $u_2(A)$; $r_2(B)$;		125
	B := B*2;		
	$w_2(B)$; $u_2(B)$;		250

Elérhető egy konzisztens adatbázis-állapotot eredményező ütemezés.

A tranzakciók kétfázisúak!

A kétfázisú zárolás (two-phase locking, 2PL)

Tétel: **Konzisztens, kétfázisú zárolású tranzakciók bármely S jogszerű ütemezését át lehet alakítani konfliktusekvivalens soros ütemezéssé.**

Bizonyítás: **S-ben részt vevő tranzakciók száma (n) szerinti indukcióval.**

Megjegyzés:

A konfliktusekvivalencia csak az olvasási és írási műveletekre vonatkozik: Amikor felcseréljük az olvasások és írások sorrendjét, akkor figyelmen kívül hagyjuk a zárolási és zárfeloldási műveleteket. Amikor megkaptuk az olvasási és írási műveletek sorrendjét, akkor úgy helyezzük el köréjük a zárolási és zárfeloldási műveleteket, ahogyan azt a különböző tranzakciók megkövetelik. Mivel minden tranzakció felszabadítja az összes zárolást a tranzakció befejezése előtt, tudjuk, hogy a soros ütemezés jogszerű lesz.

A kétfázisú zárolás

Tétel: Konzisztens, kétfázisú zárolású tranzakciók bármely S jogszerű ütemezését át lehet alakítani konfliktusekvivalens soros ütemezéssé.

Bizonyítás:

Alapeset: Ha n = 1, azaz egy tranzakcióból áll az ütemezés, akkor az már önmagában soros, tehát biztosan konfliktus-sorbarendezhető.

Indukció: Legyen S a $T_1, T_2, \ldots, T_n$ n darab konzisztens, kétfázisú zárolású tranzakció műveleteiből álló ütemezés, és legyen T_i az a tranzakció, amelyik a teljes S ütemezésben a **legelső zárfeloldási műveletet** végzi, mondjuk $u_i(X)$-t.

Azt állítjuk, hogy T_i **összes olvasási és írási műveletét az ütemezés legelejére tudjuk vinni** anélkül, hogy konfliktusműveleteken kellene áthaladnunk.

A kétfázisú zárolás

Vegyünk egy konfliktusos párt, $w_i(Y)$-t és $w_j(Y)$-t. (Hasonlóan látható be más konfliktusos párra is.)

Tekintsük T_i valamelyik műveletét, mondjuk $w_i(Y)$-t. Megelőzheti-e ezt S-ben valamely konfliktusművelet, például $w_j(Y)$? Ha így lenne, akkor az S ütemezésben az $u_j(Y)$ és az $l_i(Y)$ műveletek az alábbi módon helyezkednének el:

$$\ldots; w_j(Y); \ldots; u_j(Y); \ldots; l_i(Y); \ldots; w_i(Y); \ldots$$

Mivel T_i az első, amelyik zárat old fel, így S-ben $u_i(X)$ megelőzi $u_j(Y)$-t, vagyis S a következőképpen néz ki:

$$\ldots; w_j(Y); \ldots; u_i(X); \ldots; u_j(Y); \ldots; l_i(Y); \ldots; w_i(Y); \ldots$$

Az $u_i(X)$ művelet állhat $w_j(Y)$ előtt is. Mindkét esetben $u_i(X)$ $l_i(Y)$ előtt van, ami azt jelenti, hogy T_i nem kétfázisú zárolású, amint azt feltételeztük.

A kétfázisú zárolás

Bebizonyítottuk, hogy S legelejére lehet vinni T_i összes műveletét konfliktusmentes olvasási és írási műveletekből álló műveletpárok cseréjével.

Ezután elhelyezhetjük T_i zárolási és zárfeloldási műveleteit. Így S a következő alakba írható át:

(T_i műveletei) (a többi n-1 tranzakció műveletei)

Az n-1 tranzakcióból álló második rész szintén konzisztens 2PL tranzakciókból álló jogszerű ütemezés, így alkalmazhatjuk rá az indukciós feltevést. Átalakítjuk a második részt konfliktusekvivalens soros ütemezéssé, így a teljes S konfliktus-sorbarendezhetővé vált.

Q.E.D.

A holtpont kockázata

T_1: $l_1(A); r_1(A); A := A+100; w_1(A); l_1(B);$
$u_1(A); r_1(B); B := B+100; w_1(B); u_1(B);$

T_2: $l_2(B); r_2(B); B := B*2; w_2(B); l_2(A);$
$u_2(B); r_2(A); A := A*2; w_2(A); u_2(A);$

T_1	T_2	A	B
$l_1(A)$; $r_1(A)$;		25	
	$l_2(B)$; $r_2(B)$;		25
A := A+100;			
	B := B*2;		
$w_1(A)$;		125	
	$w_2(B)$;		50
$l_1(B)$; elutasítva	$l_2(A)$; elutasítva		

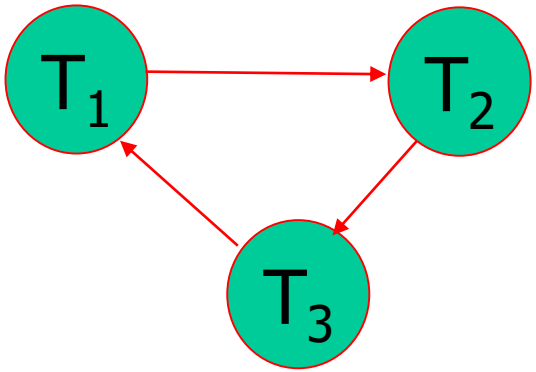
Egyik tranzakció sem folytatódhat, hanem örökké várakozniuk kell.

Nem tudjuk mind a két tranzakciót folytatni, ugyanis ha így lenne, akkor az adatbázis végső állapotában nem lehetne A=B.

Holtpont felismerése

- **A felismerésben segít a zárkérések sorozatához tartozó várakozási gráf: csúcsai a tranzakciók és akkor van él T_i-ből T_j-be, ha T_i vár egy olyan zár elengedésére, amit T_j tart éppen.**
- A várakozási gráf változik az ütemezés során, ahogy újabb zárkérések érkeznek vagy zárelengedések történnek, vagy az ütemező abortáltat egy tranzakciót.
- $l_1(A)$; $l_2(B)$; $l_3(C)$; $l_1(B)$; $l_2(C)$; $l_3(A)$

Az ütemezésnek megfelelő várakozási gráf:



Holtpont felismerése

- **Tétel**. ***Az ütemezés során egy adott pillanatban pontosan akkor nincs holtpont, ha az adott pillanathoz tartozó várakozási gráfban nincs irányított kör.***

  Bizonyítás: Ha van irányított kör a várakozási gráfban, akkor a körbeli tranzakciók egyike se tud lefutni, mert vár a mellette levőre. Ez holtpont.

  Ha a gráfban nincs irányított kör, akkor van topológikus rendezése a tranzakcióknak és ebben a sorrendben le tudnak futni a tranzakciók. (Az első nem vár senkire, mert nem megy belőle ki él, így lefuthat; ezután már a másodikba se megy él, az is lefuthat, és így tovább). Q.E.D.

Megoldások holtpont ellen

1. Rajzoljuk folyamatosan a várakozási gráfot és ha holtpont alakul ki, akkor ABORT-áljuk az egyik olyan tranzakciót, aki benne van a kialakult irányított körben.

   Ez egy megengedő megoldás (optimista), hagyja az ütemező, hogy mindenki úgy kérjen zárat, ahogy csak akar, de ha baj van, akkor erőszakosan beavatkozik. Az előző példa esetében mondjuk kilövi T_2-t, ettől lefuthat T_1, majd T_2 is.

2. Pesszimista hozzáállás: ha hagyjuk, hogy mindenki össze-vissza kérjen zárat, abból baj lehet. Előzzük inkább meg a holtpont kialakulását valahogyan. Lehetőségek:

(a) Minden egyes tranzakció előre elkéri az összes zárat, ami neki kelleni fog. Ha nem kapja meg az összeset, akkor egyet se kér el, el se indul.

Ilyenkor biztos nem lesz holtpont, mert ha valaki megkap egy zárat, akkor le is tud futni, nem akad el. Az csak a baj ezzel, hogy előre kell mindent tudni.

(b) Feltesszük, hogy van egy sorrend az adategységeken és minden egyes tranzakció csak eszerint a sorrend szerint növekvően kérhet újabb zárakat. Itt lehet, hogy lesz várakozás, de holtpont biztos nem lesz. Miért?

Megoldások holtpont ellen

Tegyük fel, hogy T_1,..., T_n irányított kört alkot, ahol T_i vár T_{i+1}-re az A_i adatelem miatt.

Ha mindegyik tranzakció betartotta, hogy egyre nagyobb indexű adatelemre kért zárat,

akkor $A1 < A2 < A3 < An < A1$ áll fenn, ami ellentmondás.

Tehát ez a protokoll is megelőzi a holtpontot, de itt is előre kell tudni, hogy milyen zárakat fog kérni egy tranzakció.

Még egy módszer, ami szintén optimista, mint az első:

Időkorlát alkalmazása: ha egy tranzakció kezdete óta túl sok idő telt el, akkor ABORT-áljuk.

Ehhez az kell, hogy ezt az időkorlátot jól tudjuk megválasztani.

Éhezés

Másik probléma, ami zárakkal kapcsolatban előfordulhat:

éhezés: többen várnak ugyanarra a zárra, de amikor felszabadul mindig elviszi valaki a tranzakció orra elől.

Megoldás: adategységenként FIFO listában tartani a várakozókat, azaz mindig a legrégebben várakozónak adjuk oda a zárolási lehetőséget.

Különböző zármódú zárolási rendszerek

Probléma: a T tranzakciónak akkor is zárolnia kell az X adatbáziselemet, ha csak olvasni akarja X-et, írni nem.

- Ha nem zárolnánk, akkor esetleg egy másik tranzakció azalatt írna X-be új értéket, mialatt T aktív, ami nem sorba rendezhető viselkedést okoz.
- Másrészről pedig miért is ne olvashatná több tranzakció egyidejűleg X értékét mindaddig, amíg egyiknek sincs engedélyezve, hogy írja.

Osztott és kizárólagos zárak

Mivel ugyanannak az adatbáziselemnek két olvasási művelete nem eredményez konfliktust, így ahhoz, hogy az olvasási műveleteket egy bizonyos sorrendbe soroljuk, nincs szükség zárolásra.

Viszont szükséges azt az elemet is zárolni, amelyet olvasunk, mert ennek az elemnek az írását nem szabad közben megengednünk.

Az íráshoz szükséges zár viszont „erősebb”, mint az olvasáshoz szükséges zár, mivel ennek mind az olvasásokat, mind az írásokat meg kell akadályoznia.

Osztott és kizárólagos zárak

A legelterjedtebb zárolási séma két különböző zárat alkalmaz: az *osztott zárakat* (shared locks) vagy *olvasási zárakat*, és a *kizárólagos zárakat* (exclusive locks) vagy *írási zárakat*.

Tetszőleges `X` adatbáziselemet vagy egyszer lehet zárolni kizárólagosan, vagy akárhányszor lehet zárolni osztottan, ha még nincs kizárólagosan zárolva.

Amikor írni akarjuk `X`-et, akkor `X`-en kizárólagos zárral kell rendelkeznünk, de ha csak olvasni akarjuk, akkor `X`-en akár osztott, akár kizárólagos zár megfelel.

Feltételezzük, hogy ha olvasni akarjuk `X`-et, de írni nem, akkor előnyben részesítjük az osztott zárolást.

Osztott és kizárólagos zárak

Az $sl_i(X)$ jelölést használjuk arra, hogy a T_i tranzakció osztott zárat kér az X adatbáziselemre, az $xl_i(X)$ jelölést pedig arra, hogy a T_i kizárólagos zárat kér X-re.

Továbbra is $u_i(X)$-szel jelöljük, hogy T_i feloldja X zárását, vagyis felszabadítja X-et minden zár alól.

A tranzakciók konzisztenciája, a tranzakciók 2PL feltétele és az ütemezések jogszerűsége

1. ***Tranzakciók konzisztenciája:*** **Nem írhatunk kizárólagos zár fenntartása nélkül, és nem olvashatunk valamilyen zár fenntartása nélkül.**

Pontosabban fogalmazva: bármely T_i tranzakcióban

a) az $r_i(X)$ olvasási műveletet meg kell, hogy előzze egy $sl_i(X)$ vagy egy $xl_i(X)$ úgy, hogy közben nincs $u_i(X)$;

$sl_i(X) \ldots r_i(X)$ vagy $xl_i(X) \ldots r_i(X)$

b) a $w_i(X)$ írási műveletet meg kell, hogy előzze egy $xl_i(X)$ úgy, hogy közben nincs $u_i(X)$.

$xl_i(X) \ldots w_i(X)$

Minden zárolást követnie kell egy ugyanannak az elemnek a zárolását feloldó műveletnek.

A tranzakciók konzisztenciája, a tranzakciók 2PL feltétele és az ütemezések jogszerűsége

2. *Tranzakciók kétfázisú zárolása:* **A zárolásoknak meg kell előzniük a zárak feloldását.**

Pontosabban fogalmazva: bármely T_i kétfázisú zárolású tranzakcióban egyetlen $sl_i(X)$ vagy $xl_i(X)$ műveletet sem előzhet meg egyetlen $u_i(Y)$ művelet sem semmilyen Y-ra.

$$sl_i(X) \ldots u_i(Y)$$

vagy $$xl_i(X) \ldots u_i(Y)$$

A tranzakciók konzisztenciája, a tranzakciók 2PL feltétele és az ütemezések jogszerűsége

3. *Az ütemezések jogszerűsége:* **Egy elemet vagy egyetlen tranzakció zárol kizárólagosan, vagy több is zárolhatja osztottan, de a kettő egyszerre nem lehet. Pontosabban fogalmazva:**

a) **Ha $xl_i(X)$ szerepel egy ütemezésben, akkor ezután nem következhet $xl_j(X)$ vagy $sl_j(X)$ valamely i-től különböző j-re anélkül, hogy közben ne szerepelne $u_i(X)$.**

$$xl_i(X) \ldots u_i(X) \ldots xl_j(X)$$

$$\text{vagy } xl_i(X) \ldots u_i(X) \ldots sl_j(X)$$

b) **Ha $sl_i(X)$ szerepel egy ütemezésben, akkor ezután nem következhet $xl_j(X)$ valamely i-től különböző j-re anélkül, hogy közben ne szerepelne $u_i(X)$.**

$$sl_i(X) \ldots u_i(X) \ldots xl_j(X)$$

A tranzakciók konzisztenciája, a tranzakciók 2PL feltétele és az ütemezések jogszerűsége

Megjegyzések:

1. **Az engedélyezett, hogy egy tranzakció ugyanazon elemre kérjen és tartson mind osztott, mind kizárólagos zárat, feltéve, hogy ezzel nem kerül konfliktusba más tranzakciók zárolásaival.** $sl_i(X) \ldots xl_i(X)$

2. **Ha a tranzakciók előre tudnák, milyen zárakra lesz szükségük, akkor biztosan csak a kizárólagos zárolást kérnék, de ha nem láthatók előre a zárolási igények, lehetséges, hogy egy tranzakció osztott és kizárólagos zárakat is kér különböző időpontokban.**

A tranzakciók konzisztenciája, a tranzakciók 2PL feltétele és az ütemezések jogszerűsége

T_1: $sl_1(A); r_1(A); xl_1(B); r_1(B); w_1(B); u_1(A); u_1(B);$

T_2: $sl_2(A); r_2(A); sl_2(B); r_2(B); u_2(A); u_2(B);$

T_1	T_2
$sl_1(A); r_1(A);$	
	$sl_2(A);$
	$r_2(A);$
	$sl_2(B);$
	$r_2(B);$
$xl_1(B);$ elutasítva	
	$u_2(A);$
	$u_2(B);$
$xl_1(B); r_1(B); w_1(B);$	
$u_1(A); u_1(B);$	

Az ütemezés konfliktus-sorbarendezhető.

A konfliktusekvivalens soros sorrend a (T_2, T_1), hiába kezdődött T_1 előbb.

A tranzakciók konzisztenciája, a tranzakciók 2PL feltétele és az ütemezések jogszerűsége

Tétel: Konzisztens 2PL tranzakciók jogszerű ütemezése konfliktus-sorbarendezhető.

Bizonyítás: Ugyanazok a meggondolások alkalmazhatók az osztott és kizárólagos zárakra is, mint korábban. Q.E.D.

Megjegyzés: Az ábrán T_2 előbb old fel zárat, mint T_1, így azt várjuk, hogy T_2 megelőzi T_1-et a soros sorrendben. Megvizsgálva az olvasási és írási műveleteket, észrevehető, hogy $r_1(A)$-t T_2 összes műveletén át ugyan hátra tudjuk cserélgetni, de $w_1(B)$-t nem tudjuk $r_2(B)$ elé vinni, ami pedig szükséges lenne ahhoz, hogy T_1 megelőzze T_2-t egy konfliktusekvivalens soros ütemezésben.

Kompatibilitási mátrixok

- A *kompatibilitási mátrix* minden egyes zármódhoz rendelkezik egy-egy sorral és egy-egy oszloppal.
- A sorok egy másik tranzakció által az X elemre elhelyezett záraknak, az oszlopok pedig az X-re kért zármódoknak felelnek meg.
- **A kompatibilitási mátrix használatának szabálya:**

  Egy A adatbáziselemre C módú zárat akkor és csak akkor engedélyezhetünk, ha a táblázat minden olyan R sorára, amelyre más tranzakció már zárolta A-t R módban, a C oszlopban „igen” szerepel.

- **Az osztott (S) és kizárólagos (X) zárak kompatibilitási mátrixa:**

Megkaphatjuk-e ezt a típusú zárat?

Ha ilyen zár van már kiadva

	S	X
S	igen	nem
X	nem	nem

Összefoglalás

- Konfliktus-sorbarendezhetőség tesztelése megelőzési gráf alapján
- Passzív módszer
- Aktív módszer
  - Zárolás
  - Időbélyezés
  - Érvényesítés
- Zárolási ütemező
- Tranzakció (konzisztens, 2PL) + Ütemezés (jogszerű) -> sorbarendezhetőség
- Holtpont (felismerés várakozási gráffal, további megoldások), Kiéheztetés (FIFO lista)
- Osztott és kizárólagos zárak
- Tranzakció (konzisztens, 2PL) + Ütemezés (jogszerű) -> sorbarendezhetőség
- Kompatibilitási mátrix