

Deadlock :

— başka kaynakları belirleyen process'ler arasında deadlock olur.

* İşletim sistemleri genellikle deadlock önleme mekanizmaları saflamazlar.

Program geliştiricilerin deadlock olusmayarak şekilde program geliştirmesi gereklidir.

Kaynaklar (Resources) :

CPU cycle, dosyalar, I/O cihazları (kızılı, DVD sürücü) gibi, sembolü R

Bir process bir kaynağı aşağıdaki sırayla kullanır :

- ① **Request :** Process kaynağı istek yapsa, kaynak kullanılabilir değilse bekle.
- ② **Use :** Kaynak kullanılabilir ise process kaynak üzerindeki işlevsili gerçekleştirir.
- ③ **Release :** Process kaynağı serbest bırakır.

* Request ve release çağrıları, cihaz için request() ve release(), dosya için open() ve close(), hafıza için allocate() ve free() sistem çağrıları şeklinde olabilir.

* Semforlar için wait() ve signal()

Mutex locks için acquire() ve released()

4 Durum Aynı Anda Oluştuğunda Deadlock ortaya Çıkarılır:

① Mutual Exclusion

② Hold and Wait:

Bir process bir kaynağın tutarken başka bir kaynağın da bekler durumunda.

③ No Preemption

④ Circular Wait

$P_0 \rightarrow P_1 \rightarrow P_2 \rightarrow P_3 \rightarrow P_0$

Kaynak Tahsisi Grafi (Resource-Allocation Graph):

Deadlock'lar tek gözlü graf kullanılarak tanımlanabilir.

(V) Vertices veya Düğümler:

$P = \{P_1, P_2, \dots, P_N\}$ aktif process'leri gösterir.

$R = \{R^1, R^2, \dots, R^N\}$ sistemdeki tüm kaynakları gösterir.

$P_i \rightarrow R_j$: P_i process'inin R_j kaynağına istek yaptığı ve bittiği ifade eder.

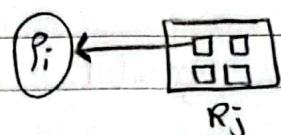
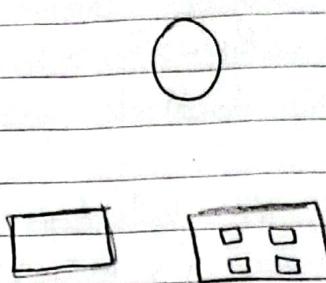
$R_j \rightarrow P_i$: R_j kaynağının P_i process'ine atandığı ifade eder.

$P_i \rightarrow R_j$: istek kenarı, request edge

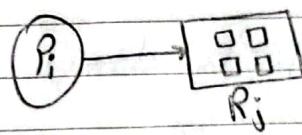
$R_j \rightarrow P_i$: Atama kenarı, assignment edge

* Bir process daire ile, her bir kaynak dikdörtgenle gösterilir.

* Bir kaynaktan birden fazla ömek varsa dikdörtgen içerişinde her ömek ayrı nokta ile gösterilir



Atama Kenarı



İstek kenarı

* Eğer grafla kenarlar döngü oluşturuyorsa, sistemdeki her bir process'in deadlock olmadığını gösterir.

* Grafla bir döngü içeriyorsa, bir deadlock olabilir.

* Döngü yalnızca her biri tek bir ömeye sahip olan bir dizî kaynak türü içeriyorsa bir deadlock olmaz. Döngüde yer alan her process deadlock olmaz.

* Her kaynak türünün birbirin ögesi varsa, bir döngünün mutlaka bir deadlock oluşturacağı anlamına gelmez.

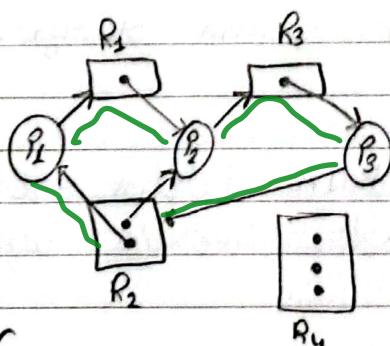
Örnek:

Döngüler:

$$P_1 \rightarrow R_1 \rightarrow P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$$

$$P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_2$$

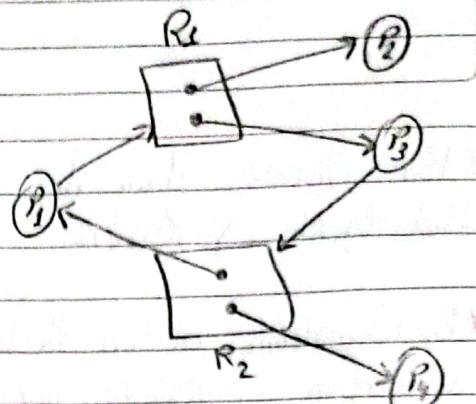
P_1, P_2 ve P_3 process'leri deadlock olmaz.



Döngü

$P_1 \rightarrow R_1 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$

Anahtar deadlock yoktur.



P_1, R_2 kullandığı kaynak, serbest bırakırsa P_3 kullanabilir, döngü kırılabilir.



* Eğer grafte döngü yoksa \rightarrow deadlock yoktur.

* Eğer grafte döngü varsa:

- Her kaynak türünün sadece bir örneği varsa deadlock olur.
- Her kaynak türünün birden fazla örneği varsa deadlock olasılığı vardır.

Deadlock Yönetimi İçin Metotlar 3 farklı yol izleyebilir:

- ① Protokol kullanılır, böylece sistem hiçbir zaman deadlock durumuna düşmez.
- ② Deadlock algılanır ve çözülür.
- ③ Deadlock problemi tamamen gözardı edilir ve sisteme deadlock hâlde zaman olmayaçak gibi davranışır.

* 3. durum işletim sistemleri tarafından yaygın kullanılır (Linux, windows) deadlock yönetimi uygulama geliştiricilere birakır.

Hold and Wait - Deadlock Önleme :

- ① Her process'in yürütülmeye başlamadan önce tüm kaynakları talep ettiği ve bunların process'e tahsis edildiği bir protokol kurulabilir.
- ② Eğer bir process kaynak kullanmıyorsa yeni kaynak için istek yapabilir.
- * Bir process kendisine tahsis edilmesi tüm kaynakları serbest bırakması gereklidir.
(Yeni veya ek bir kaynak talep etmeden önce)
- * Starvation mümkündür. bir process, sürekli olarak beklenerek zarunda kalabilir.

No Preemption - Deadlock Önleme :

- ① Bir process tahsis edilmeyen başka bir kaynak talep edilirse process'in elinde tuttuğu tüm kaynaklar serbest bırakılır.
- * Bırakılan kaynaklar, process'in beklediği kaynaklar listesine eklenir.

Baska bir process tarafından tutuluyorsa, buna process'in başka bir kaynaktan bekleyip beklemeydiğinin kontrol edilir.

Bekliyorsa, istek yapılmayan kaynak alınır yeni istek yapan process'e atanır.

Beklmiyorsa, istek yapan process bekletilir.

Circular Wait - Deadlock Problemi :

Her kaynak türüne farklı bir tam sayı atanarak tüm kaynakları sıralanır, bir process kaynak istedigi aneak artan sırada yapabilir.

* Alternatif bir protokol olacak, R_j kaynak talep edilsin

$F(R_j) \geq F(R_i)$ olaak şekilde R_j kaynağının serbest birikmesini sağlayabiliriz.

Deadlock'un Karınma - Deadlock Avoidance :

Deadlock oluşumunu engellemek için kaynakların nasıl istendiğini bilinmesi gereklidir.

* Eğer bir sistem kaynakları process'lere belirli bir sırada maksimum ihtiyacına kadar ulaşabiliyorsa ve deadlock oluşmuyorsa bu durum **safe state** olarak adlandırılır.

Eğer bir sistende **safe sequence** varsa aneak sistem **safe state** olur.

Process aneak aşağıdaki durumları sağluyorsa **Safe Sequence**,

- P_i process'inin tüm kaynak istekleri mercut boş kaynaklarla ve tüm P_j 'ler ($i < j$) tarafından tutulmuş ddu kaynaklarla varlığını abiliyorsa.

Bu durumda, P_i 'nin ihtiyaç duyduğu kaynaklar hemen mercut değilse, P_i tüm P_j 'lerin tamamlanması bekler.

- Tüm P_j 'ler tamamlandığında P_i ihtiyaç duyduğu tüm kaynakları elle edebilir.

- P_i sonra erdiğinde, $P_i + 1$ geceli kaynakları elle edebilir ve bu böyle devam eder. Böyle bir sıra yoksa.

Sistem durumunun unsafe olduğu söylenilir.

Banker Algoritması - Deadlock'tan Kafınma 3

Aynı kaynakları birden fazla olan sistemlerde banker algoritması kullanılır.

Bir kullanıcı bir dizi kaynak talep ettiğinde, sistem bu kaynakların tamamsı elmenin sistem safe state'ye bırakıp bırakmayacağı belidir.

Safe State'ye bırakılaraksa kaynaklar tahsis edilir, aksi takdirde process başka bir process'in getirdiği kaynakları serbest bırakmasını beklemelidir.

Kullanılan Veri Yapıları:

n⁸ Process Sayısı

m⁸ Kaynak Türü Sayısı

Available:

m uzantılıka

Available[j]=k

R_j kaynaklarından k adet boşta

Max:

n⁸ m uzantılıka

Max[i,j]=k

P_i processi R_j kaynaklarından en fazla k tane istegebilir.

Allocation:

n⁸ m uzantılıka

Allocation[i,j]=k

P_i processi R_j kaynaklarından k adet kullanmaktadır.

Need:

$$Need[i,j] = Max[i,j] - Allocation[i,j]$$

n⁸ m uzantılıka

Need[i,j]=k

P_i processi R_j kaynaklarında k adet daha kullanabilir

* Single instance resource \rightarrow Kaynak tahisisi grafi
Multiple instances of resource \rightarrow Banker Algo.

Banker algoritması için Güvenlik Algoritması (Safety Algorithm):

① $Work = Available$ (m uzunlukta)

$Finish[i] = \text{false}$ for $i=0, 1, \dots, n-1$ (n uzunlukta)

② Aşağıdaki 2 koşulu sağlayan bir i processi bul:

$Finish[i] = \text{false}$

$Need_i \leq Work$ // i 'nin ihtiyaç sayısı boşta olanlardan küçük ise
 varsa 3. adıma git
 yoksa 4. adıma git

③ $Work = Work + Allocation$

$Finish[i] = \text{true}$

2. adıma git

④ $Finish[i] = \text{true}$

örnek

Sistemde P_0 dan P_4 e, 5 tane process

3 kaynak türü:

A (10 örnek) B (5 örnek) ve C (7 örnek)

Safe Sequence: $\langle P_1, P_3, P_4, P_0, P_2 \rangle$

Tümunda sistem bu şekilde

	Alloc	Max	Need	Work	
	A B C	A B C	A B C	ABC	Aval
P_0	0 0 0	7 5 3	3 4 3	5 3 2	7 8 5
P_1	2 0 0	3 2 2	1 2 2		
P_2	0 8 0	9 0 8	6 0 0	1 0 5	7
P_3	0 2 1	2 2 2	0 1 0	7 4 8	
P_4	0 0 2	4 3 3	1 3 *	7 4 5	

① $Need = Max - Alloc$

② $Aval \rightarrow Work$

③ P_0 need, work ile sağlanır

④ P_1 need, work ile sağlanır

Alloc, Max, Need sıfırlanır
 $Work = Work + Alloc(P_1)$

$Finish[1] = \text{true}$

⑤ P_2 need, work ile sağlanır

⑦ P_4 need, work ile sağlanır

Alloc, Max, Need sıfırlanır

$Work = Work + Alloc(P_4)$

$Finish[4] = \text{true}$

tüm i'ler için true old.
 Sistem güvenli durumda.

⑥ P_3 need, work ile sağlanır

Alloc, Max, Need sıfırlanır

$Work = Work + Alloc(P_3)$

$Finish[3] = \text{true}$

Kaynak İstek Algoritması - (Resource - Request) :

İsteklerin onaylananın güvenli ~~durum~~ durumu bozulmamasını anlamak için kullanılıyor.

Request; : P_i Prosesi için istek vektörü

Request; $[ij]$: P_i Prosesi, R_j kaynak türünün k örneğini isteri.

P_i tarafından Kaynak talebi yapıldığında algoritmanın adımları?

- ① $\text{Request}_i \leq \text{Need}_i$ ise 2. adıma git
değilse maksimum talebi astır, iin hata ver.
- ② $\text{Request}_i \leq \text{Available}$ ise 3. adıma git
değilse Kaynaklar boş olmadığından P_i beklemeli.
- ③ $\text{Available} = \text{Available} - \text{Request}_i$
 $\text{Allocation}_i = \text{Allocation}_i + \text{Request}_i$
 $\text{Need}_i = \text{Need}_i - \text{Request}_i$
- ④ Yeni tablo için
banker alg. su...
Safe Olmalı!!!
Request kabul edilir.

Durum güvenli ise P_i 'ye isteği kaynaklar tahsis edilir.

Değilse P_i , Request $_i$ için beklemelidir.

Örnegin, P_1 prosesi Request $= (1, 0, 2)$ isterse

	Allocation	Max	Available		Allocation	Need	Available	
	A B C	A B C	A B C		A B C	A B C	A B C	
T ₀	0 1 0	7 5 3	3 3 2		T ₀	0 1 0	7 4 3	2 3 0
T ₁	2 0 0	3 2 2		T ₁	3 0 2	0 2 0		
T ₂	3 0 2	9 0 2		T ₂	3 0 2	6 0 0		
T ₃	2 1 1	2 2 2		T ₃	2 1 1	0 1 1		
T ₄	0 0 2	4 3 3		T ₄	0 0 2	4 3 1		

$$\textcircled{1} \quad \text{Request}_1 \leq \text{Need}_1 \Rightarrow (1, 0, 2) \leq (1, 2, 2)$$

$$\textcircled{2} \quad \text{Request}_1 \leq \text{Available} \Rightarrow (1, 0, 2) \leq (3, 3, 2)$$

Deadlock Algılama :

Algılama algoritmaları, her kaynaktan bir tanesi olması veya her kaynaktan birden fazla olması durumuna göre farklılık gösterir.

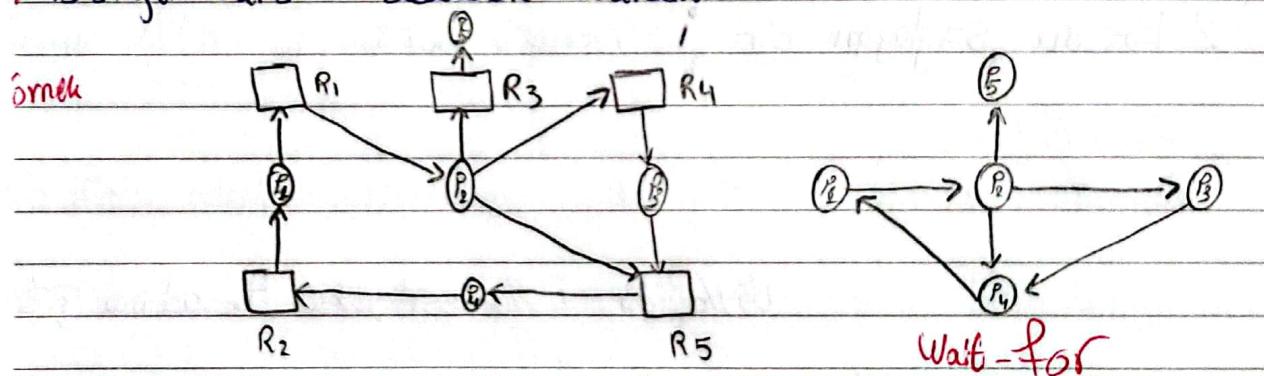
Her Kaynaktan bir örnek olması :

Wait-for grafi adı verilen bir deadlock algılama algoritması tanımlanmıştır.

- Wait-for grafında, Kaynak düğümleri kablundanak gidece process düğümleri bırakılır.

- P_i den P_j ye doğru bir kemiş $P_i \rightarrow P_j$, P_i process'inin P_j processini beklediğini gösterir.

* Döngü varsa deadlock vardır.



Her kaynakta birden fazla örnekl olmasi :

Available : $n \times m$ uzunlukunda , Böşteki kaynak sayisini tutar

Allocation : $n \times m$ uzunlukunda , Her process'in mevcut oturum kaynak sayisini tutar.

Request : $n \times m$ uzunlukunda , Her process'in mevcut istek sayisini tutar.

Request[i,j]=k : P_i processi P_j kaynaktan k tane dala istemekle

① Work = Available

Finish[i] = false for $i = 0, 1, \dots, n-1$

m uzunlukunda
 n uzunlukunda

② 2 kosulu sağlayan bir i process'i bul :

Finish[i] = false

Request[i] \leq Work // i 'nin istek sayisi boşta dontordan küçük ise

Boyle bir i processi yoksa 4. adima git

③ Work = Work + Allocation;

Finish[i] = true

2. adima git

④ Finish[i] = false // bazı i'ler için sağlanırsa sistem
// deadlock durumundadır.

Örnek

Sisteme P0 dan P4 e, 5 tane process

3 kaynaktır : A (7 örnkl), B (2 örnkl) ve C (6 örnkl)

To anında sistem

	Allocation	Request	Available
	A B C	A B C	A B C
P0	0 1 0	0 0 0	0 0 0
P1	2 0 0	2 0 2	
P2	3 0 3	0 0 0	
P3	2 1 1	1 0 0	
P4	0 0 2	0 0 2	

deadlock yolu

tümü'ler Finish[i]=true

Safe Sequence = $\langle P_0, P_2, P_3, P_1, P_4 \rangle$

Anıktan P_2 (0,0,1) isteğiini yaparsa

DEADLOCK

Request

A B C

0 0 0 ✓

2 0 2 ✗

0 0 1 ✗

1 0 0 ✗

0 0 2 ✗

work

0 1 0

deadlock oluştu

* Sık sık deadlock meydana gelirse, algılama algoritması sık sık yeniden yapılandırılmalıdır.

* Basılıca daha ucuz bir alternatif, algoritmayi tanımlı belirli aralıklarda göğürmektir.

* Bir talebin talebinin hemen kabul edilmediğ; her seferde, deadlock algılama algoritmasını çalıştırılabilir.

* Algılama algoritması bir deadlock olduğunu belirttiğinde

- Operatöre bir deadlock olusunu bildirir ve operatörün deadlock'la manuel olarak zgilenmesine izin verir.
- Sistemin deadlock durumundan otomatik çıkış kurulmasına izin verir.

Bir deadlock'tan kurtulmak için 2 seçenek var:

- ① Döngüsel beklemeyi bozmak için bir veya daha fazla process'in iptal edilmesi / sonlandırılmasıdır (abort process)
- ② Deadlock olmayan process'lerin bazı kaynakları kullanmasının kesintiliye uğratılmasıdır (Preempt edilmesi)

Process'; sonlandırarak deadlock'ları ortadan kaldırma için 2 yöntem var:

- ① Tüm deadlock durumundaki process'ler sonlandırılır. Process'lerin sonraki halebilir.
- ② Deadlock döngüsünü ortadan kalkmaya kadar her adımda bir deadlock process sonlandırılır. Her process sonlandığında deadlock cycle kontrolü yapılması gereklidir.

* Bir process'in abort edilmesi sonucunda tutsarsızlıklar ortaya çıkabilir.

* Kismi sonlandırma yöntemini kullanıysa, deadlock olmayan hangi process'in veya process'lerin sonlandırılması gerekligine karar verilmelidir.

Processlerin Sonlandırılması için faktörler

- ① Process'in önceliğinin ne olduğu
- ② Process'in ne kadar süreler yürüböldüğü ve belirlenen gerekli tamamlaması için process'in daha ne kadar süre yürübüleceği;
- ③ Process'in kaq tane ve ne tür kaynak kullandığı
- ④ Process'in tamamlanması için daha kaq kaynak ihtiyacı olduğu
- ⑤ Kaq process'in sonlandırılmas gerekligi?
- ⑥ Process'in etkileşimli mi yoksa toplu mu olduğu
(interactive) (batch)

* Proseslerin Sonlandırılmasında minimum maliyet bir kriter değildir.

Deadlock'ta boşa çıkmak için preemption gereklisi

- ① Selecting a victim
- ② Kaynağı elinden alınan process güvenli bir duruma geri alınmalı ve bu durumdan yeniden başlatılmalıdır.
- ③ Kaynağı elinden alınan process'in seçimi maliyet faktörüne göre yapıldığından aynı process sürekli olarak ~~olarak~~ seçilebilir bu da Starvation'a neden olabilir.