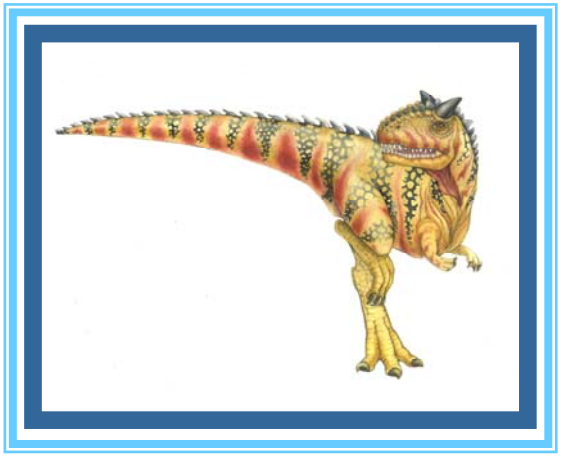
Bölüm 7: Ölümcül Kilitlenme (Deadlocks)



Bölüm 7: Ölümcül Kilitlenme

- Ölümcül Kilitlenme Problemi
- Sistem Modeli
- Ölümcül Kilitlenme Karakterizasyonu
- Ölümcül Kilitlenme Yönetim Metodları
- Ölümcül Kilitlenmeyi Önleme
- Ölümcül Kilitlenmeden Kaçınma
- Ölümcül Kilitlenme Tespiti
- Ölümcül Kilitlenmeyi Kurtarma

Bölümün Hedefleri

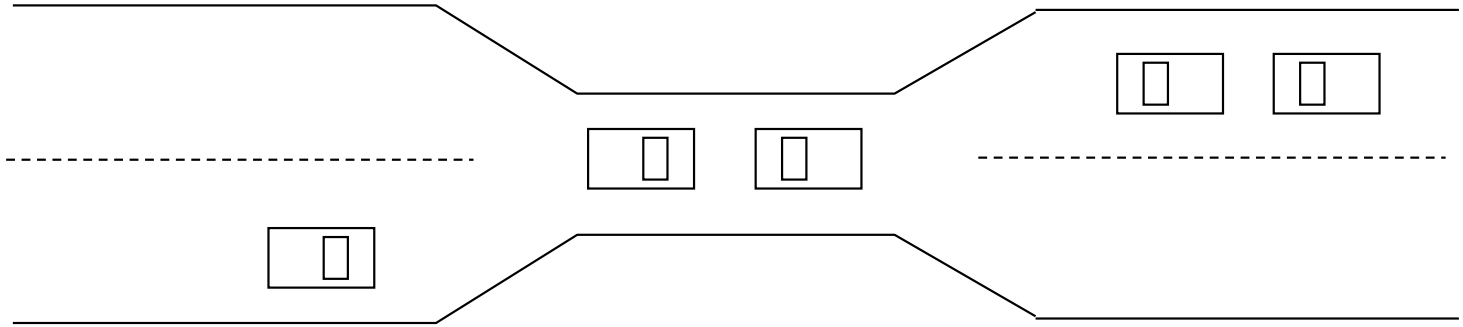
- Eşzamanlı proseslerin görevlerini tamamlamasını engelleyen ölümcül kilitlenmeyi tanımlamak
- Bir bilgisayar sisteminde ölümcül kilitlenmeleri önlemek veya kaçınmak için farklı metotlar sunmak

Ölümcül Kilitlenme Problem,

- Her biri bir kaynak tutan bir grup bloke edilmiş proses, başka prosesin tuttuğu kaynağa da sahip olmak istiyor.
- Örnek
  - Sistemde iki tane disk sürücüsü vardır
  - P_1 ve P_2 'nin her biri birer disk sürücüsü tutuyor ve her biri diğerine de ihtiyaç duyuyor.
- Örnek
  - *A* ve *B* semaforları, P_0 P_1 'i başlatır

wait (A);	wait(B)
wait (B);	wait(A)

Köprü Geçiş Örneği



- Trafik yalnızca bir yönde ilerler.
- Köprünün her bölümü kaynak olarak görülebilir.
- Eğer bir ölümcül kilitlenme oluşursa, bir aracın geri çekilmesi ile çözülebilir (kaynağı talep et ve yeniden başla)
- Eğer bir ölümcül kilitlenme oluşursa çok sayıda araba geri geri gitmek zorunda kalabilir
- Açlıktan ölme olasıdır
- Not: Birçok işletim sistemi ölümcül kilitlenmeyi önlemez
- veya ilgilenmez.

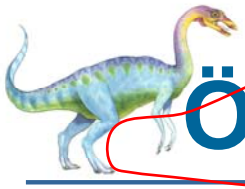
Sistem Modeli

- Kaynak tipleri R_1, R_2, . . ., R_m

  CPU çevrimleri, bellek alanları, I/O aygıtları

- Her bir kaynak tipi R_i W_i örneğine sahiptir.

- Her bir proses bir kaynağı aşağıdaki gibi kullanır:
  - **Talep et (Request)**
  - **Kullan (Use)**
  - **Serbest bırak (Release)**



Ölümcül Kilitlenme Karakterizasyonu

Ölümcül kilitlenme aşağıdaki dört durum aynı anda olursa ortaya çıkar:

- **Karşılıklı dışlama:** Bir anda sadece bir proses bir kaynağı kullanabilir.
- **Tut ve bekle:** En az bir kaynağı elinde tutan bir proses başka prosesler tarafından tutulan bir kaynağı ilave olarak edinmek ister.
- **Kesinti yok:** Bir kaynak sadece onu elinde tutan proses tarafından gönüllü olarak serbest kalır, sonra proses görevini tamamlar
- **Döngüsel bekleme:** $\{P_0, P_1, \ldots, P_n\}$ bekleyen prosesler kümesi ve P_0 , P_1 in tuttuğu bir kaynağı bekliyor; P_1, P_2 tarafından tutulan kaynağı bekliyor,

  , …, P_{n-1}, P_n in tuttuğu kaynağı bekliyor ve P_n , P_0 tarafından tutulan kaynağı bekliyor.

Kaynak-Atama Grafı

Düğümler kümesini *V* ve kenarlar kümesini *E* ile gösterelim

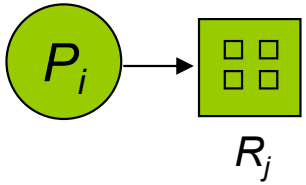
- V iki tipe ayrılır:
  - $P = \{P_1, P_2, \ldots, P_n\}$, sistemdeki tüm prosesler kümesi
  - $R = \{R_1, R_2, \ldots, R_m\}$, sistemdeki tüm kaynaklar kümesi
- **istek kenarı**– yönlü graf $P_i \rightarrow R_j$
- **atama kenarı**– yönlü graf $R_j \rightarrow P_i$

Kaynak-Atama Grafı (Devam)

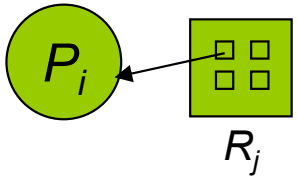
- Proses

- 4 adet örneğe sahip bir kaynak

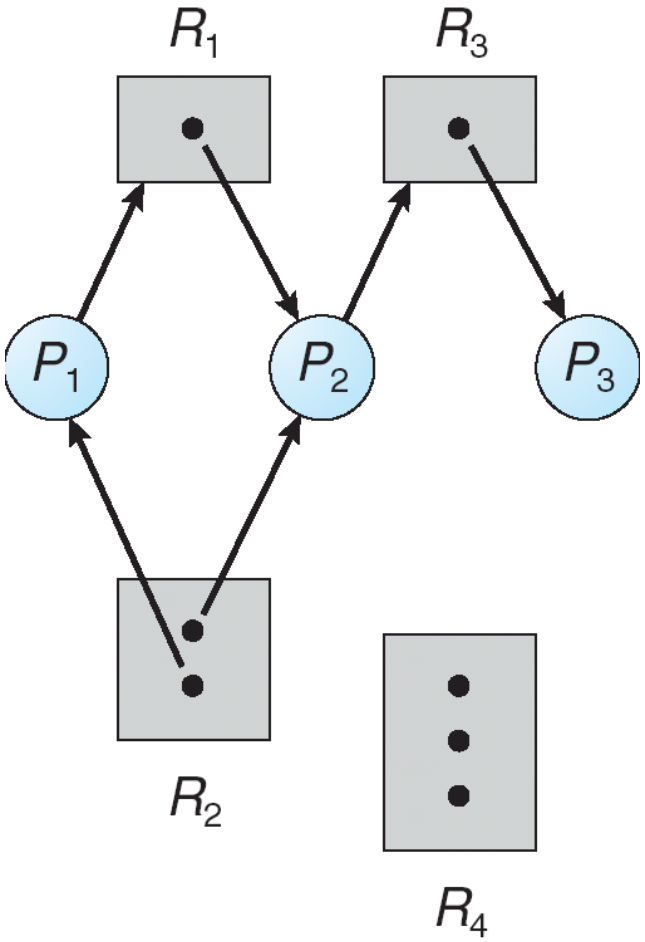
- P_i , R_j ‘den bir adet ister



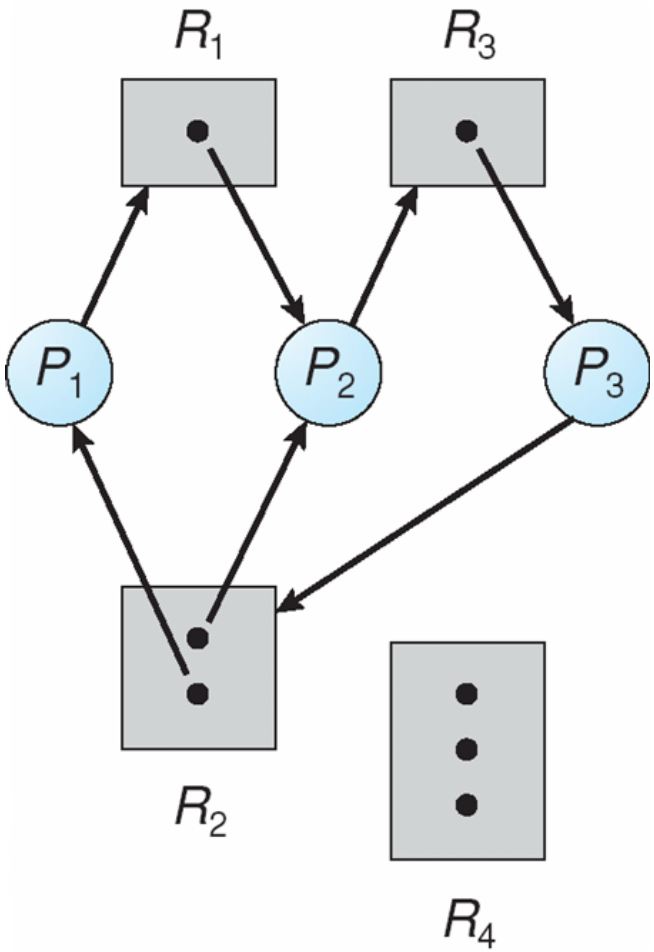
- P_i , R_j’den bir adetini elinde tutar

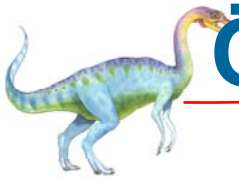


Kaynak-Atama Grafı Örneği

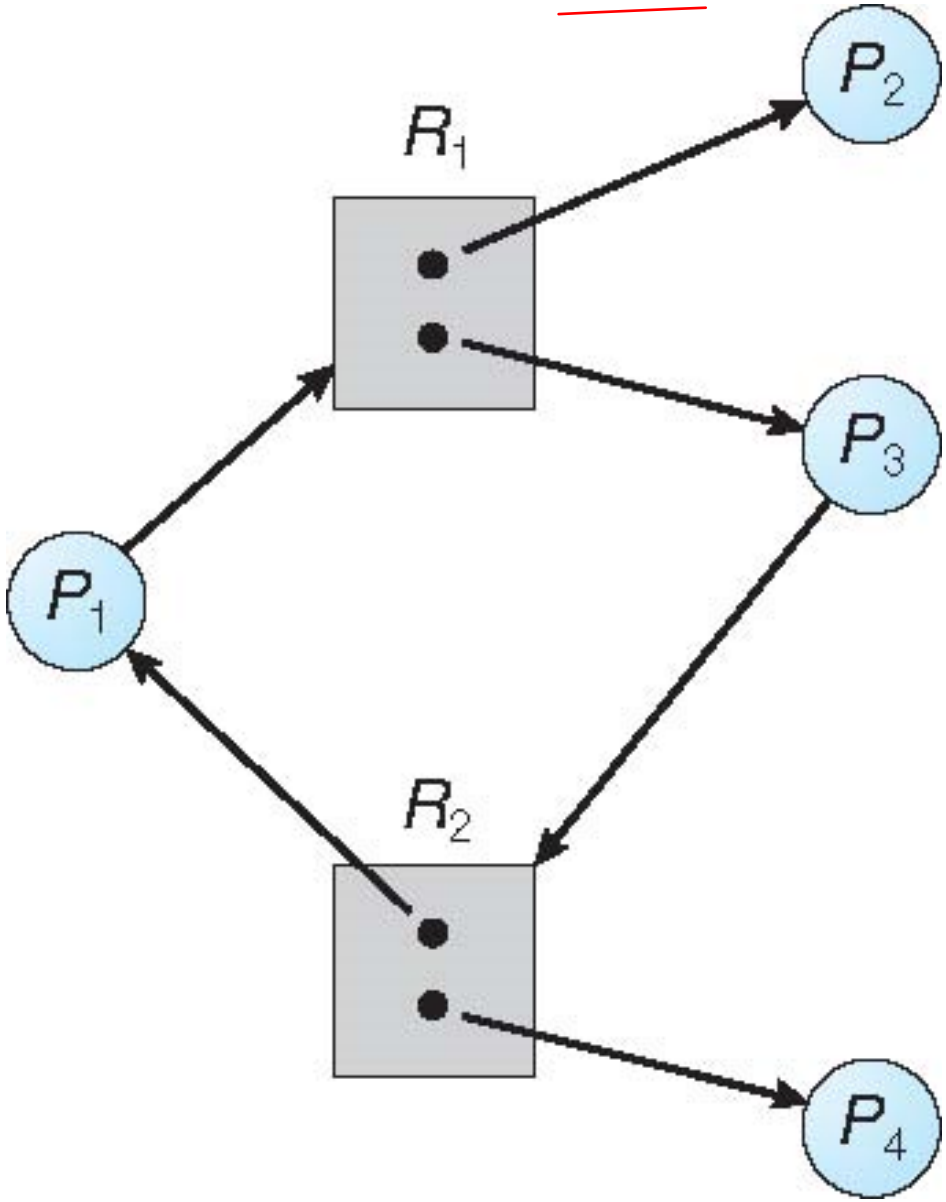


Ölümcül Kilitlenmeli Kaynak-Atama Grafı





Ölümcül Kilitlenmesiz Ancak Çevrimli Graf



Temel Bilgiler

- Eğer grafikte çevrim yoksa ⇒ ölümcül kilitlenme yoktur
- Eğer grafikte bir çevrim varsa ⇒
  - Eğer kaynak başına bir örnek varsa, ölümcül kilitlenme olur
  - Eğer kaynak başına birden fazla örnek varsa, ölümcül kilitlenme ihtimali var

Ölümcül Kilitlenme Yönetim Metodları

- Sistemin ***asla*** kilitlenme durumuna girmeyeceğini garanti et.
- Sistemin bir ölümcül kilitlenme durumuna girmesine izin ver ve daha sonra kurtar.
- Problemi yok say ve sistemde hiçbir zaman kilitlenme meydana gelmiyor gibi davran; UNIX dahil olmak üzere birçok işletim sistemi tarafından kullanılmıştır.

Ölümcül Kilitlenmeyi Önleme

Bir isteği yapılabileceği yolları engelle

- **Karşılıklı Dışlama** – paylaşılabilir kaynaklar için gerekli değildir; ancak paylaşılamaz kaynaklar için gereklidir.
- **Tut ve Bekle** – Bir işlem kaynak talep ettiğinde başka kaynak tutmadığı garanti edilmeli
  - Prosesin çalışmaya başlamadan önce kaynaklara istek yapmasını ve almasını şart koş yada prosesin sadece boştayken kaynak talep etmesine izin ver
  - Düşük kaynak kullanımı; açlıktan ölme olabilir.

Deadlock Önleme (Devam)

- **Kesinti Yok**–
  - Eğer bir kaç kaynağı tutan bir proses paylaşılamayan başka bir kaynağı isterse, tutulan tüm kaynaklar serbest kalır.
  - Serbest kalan kaynaklar bekleyen proseslerin kullanımı için listeye alınır.
  - Eski kaynaklarını geri almak isteyen ve yeni taleplerini almak proses yeniden başlatılır.
- **Çevrimsel bekleme** – Tüm kaynak türlerinin sıralanmasını ve her bir prosesin artan bir sırada kaynakları istemesini şart koş.

Ölümcül Kilitlenmeden Kaçınma

Sistemin ilave ön bilgiye sahip olmasını gerektirir

- Basit ve kullanışlı bir model, her prosesin ihtiyaç duyulabileceği her tipteki maksimum kaynak istek sayısını bildirmesini gerektirir.
- Ölümcül kilitlenmeden kaçınma algoritması dinamik olarak çevrimsel-bekleme şartının olmamasını sağlamak için kaynak-atama durumunu inceler.
- Kaynak-atama durumu, boşta ve atanmış kaynak ve proseslerin maksimum talepleri sayısıyla tanımlanır

Güvenli Durum

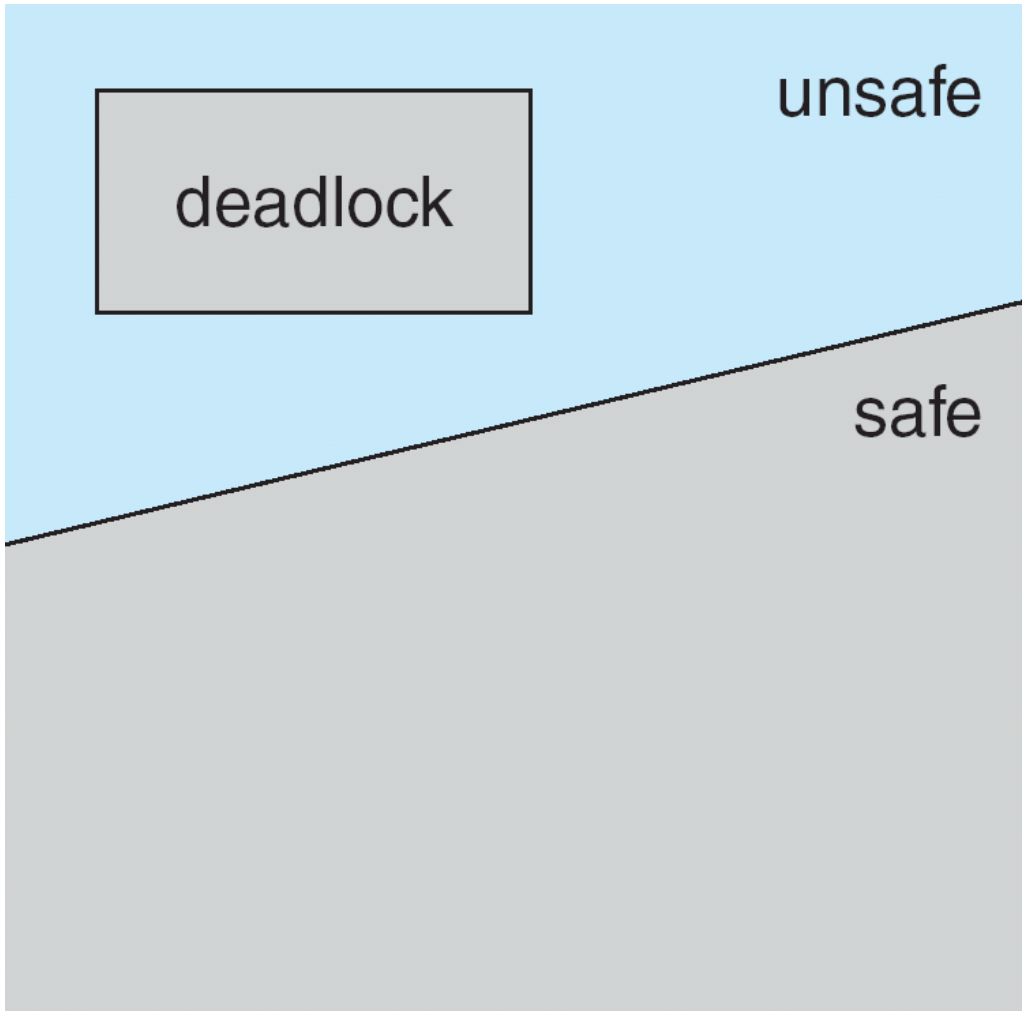
- Bir proses, boşta bir kaynağı talep ettiğinde; bu talebin yerine getirilmesinin sistemi güvenli durumdan çıkarıp çıkarmayacağını işletim sistemi karar vermelidir.

- <P_1, P_2, …, P_n> sistemdeki sıralanmış tüm prosesleri göstermek üzere herbir P_i için, P_i nin talep ettiği kaynaklar mevcut boşta kaynaklar + j < i olmak üzere tüm P_j ler tarafından tutulan kaynaklar ile sağlanıyorsa sistem güvenli durumdadır.
- Yani:
  - Eğer P_i 'nin ihtiyaç duyduğu kaynak o an için kullanılabilir değilse P_i , tüm P_j ler tamamlanana kadar bekleyebilir.
  - P_j tamamlandığında, P_i ihtiyaç duyduğu kaynakları alıp çalışabilir, daha sonra aldığı kaynakları iade edip sonlanabilir.
  - P_i sonlandığında, P_{i+1} ihtiyaç duyduğu kaynakları alıp benzer adımları gerçekleştirebilir.

Temel Bilgiler

- Eğer sistem güvenli durumdaysa ⇒ kilitlenme yok.
- Eğer sistem güvensiz durumdaysa ⇒ kilitlenme olabilir.
- Kaçınma ⇒ Sistemin asla güvensiz duruma girmemesini sağlayın.

Güvenli, Güvensiz, Ölümcül Kilitlenme Durumu



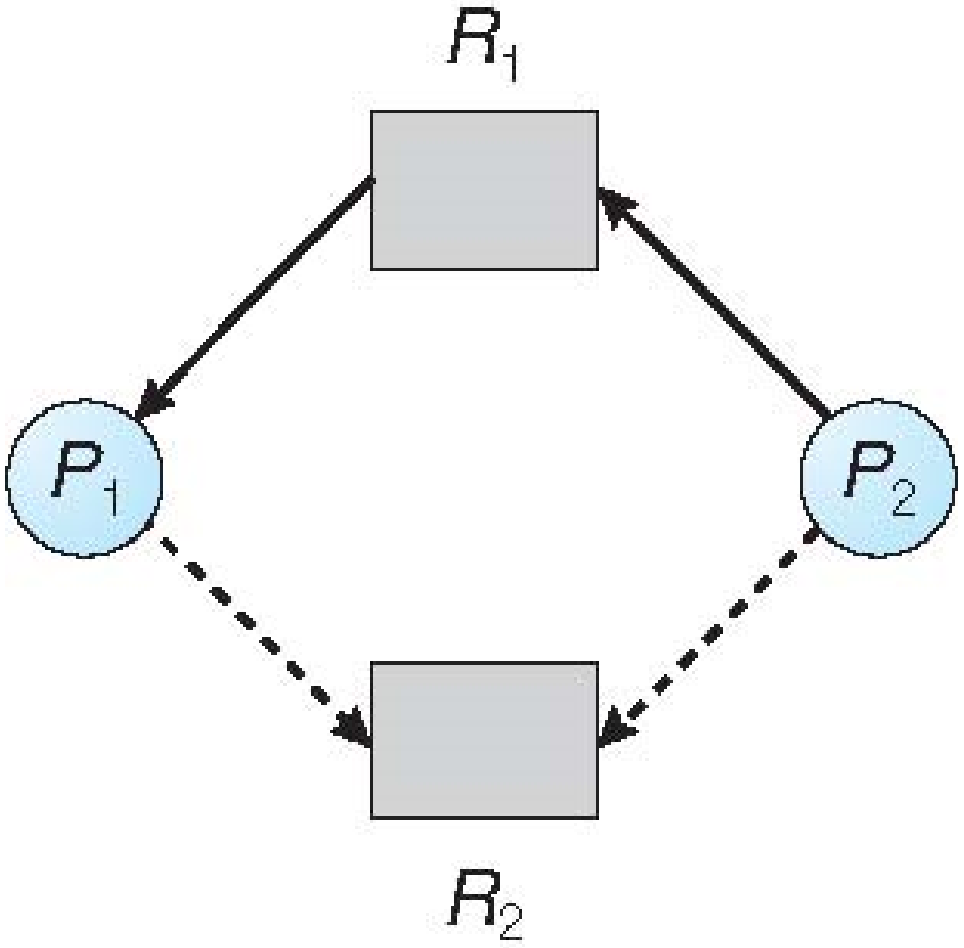
Kaçınma Algoritmaları

- Her bir kaynağın tek örneği mevcutsa:
  - Kaynak-atama grafını kullan.
- Her bir kaynaktan birden fazla mevcutsa:
  - Banker algoritmasını kullan.

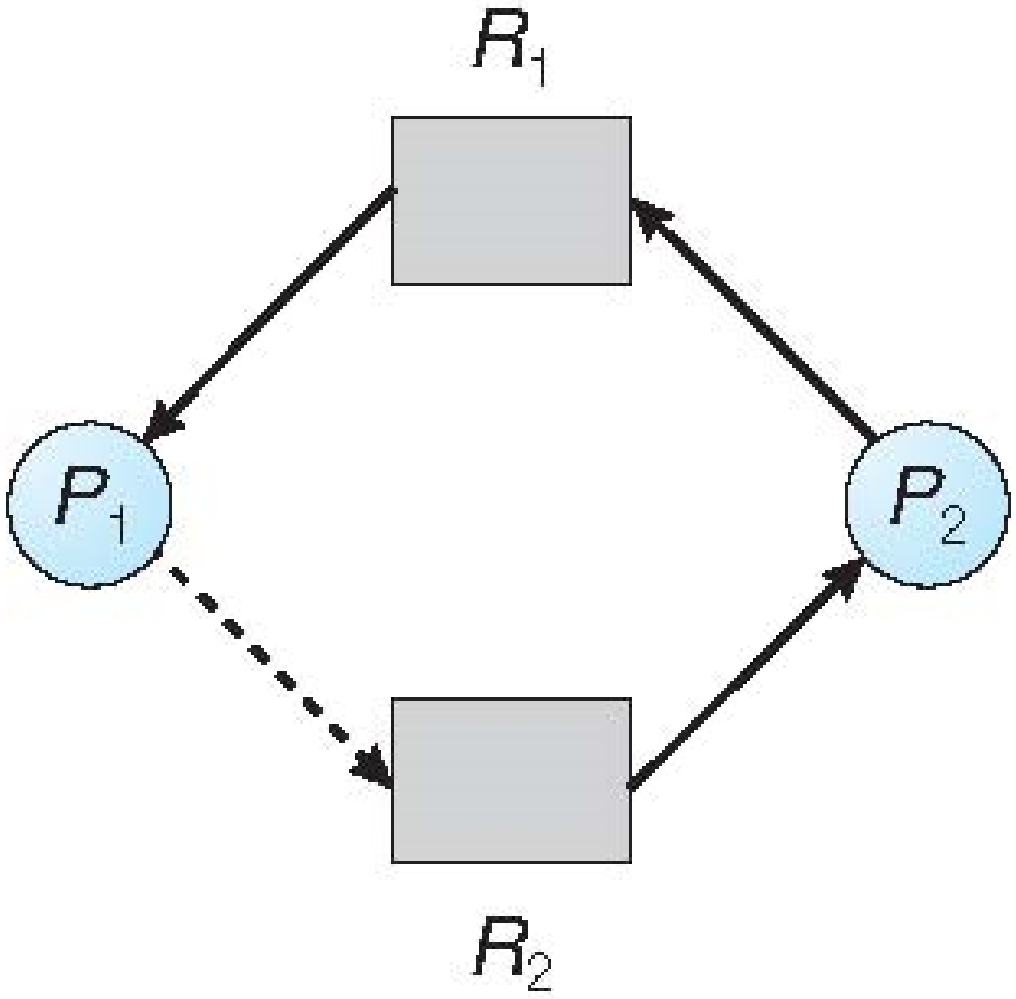
Kaynak-Atama Grafı Şeması

- **Talep kenarı** $P_i \rightarrow R_j$: P_j prosesi R_j kaynağını talep edebilir; kesik çizgiyle gösterilir
- Bir proses bir kaynağı isterse talep kenarı istek kenarına dönüşür
- Kaynak prosese tahsis edildiğinde istek kenarı atama kenarına dönüşür
- Bir kaynak proses tarafından serbest bırakılırsa atama kenarı talep kenarına
- Kaynaklar sistemde önceden talep edilmelidir

Kaynak-Atama Grafı



Kaynak-Atama Grafında Güvensiz Durum



Kaynak-Atama Grafı Algoritması

- Varsayalım ki P_i process'i, bir R_j kaynağını talep etsin.
- İstek sadece istek kenarının atama kenarına dönüşmesinin bir çevrim oluşturmadığında yerine getirilir

Banker Algoritması

- Birden çok kaynak örneği
- Her bir proses maksimum isteğini önceden deklare etmelidir
- Bir process, bir kaynak talep ettiğinde beklemesi gerekebilir.
- Bir process, talep ettiği kaynakların tümünü aldığında belirli bir süre içinde aldığı kaynakları geri vermelidir.

Banker Algoritması Veri Yapıları

n = proses sayısı, ve *m* = kaynak türü sayısı.

- **Boşta**: *m* uzunluğunda bir vektör. Eğer boşta [*j*] = *k ise*, R_j kaynak tipinin k tane kullanılabilir örneği vardır.
- **Maksimum İstek Matrisi**: *n x m* boyutunda bir matris. Eğer *Max* [*i,j*] = *k* ise, P_i prosesi R_j kaynak tipinden en fazla k tane örnek talep edebilir.
- **Atanmış Matrisi:** *n* x *m* boyutunda. Eğer Atama[*i,j*] = *k* ise P_i prosesi k tane R_j örneğini almış durumdadır.
- **İhtiyaç Matrisi**: *n* x *m* boyutunda. Eğer *İhtiyaç*[*i,j*] = *k*, ise P_i prosesi görevini tamamlamak için ilave *k* adet R_j örneğine ihtiyaç duymaktadır.

İhtiyaç [*i,j*] = *Max*[*i,j*] – *Atama*[*i,j*]

Güvenlik Algoritması

1. *Çalışan* ve *Tamamlanmış* sırasıyla *m* ve *n* büyüklüklerinde iki vektör olsun. Başlangıçta:

   Çalışan = *boş*

   Tamamlanmış [*i*] = *false* , *i* = 0, 1, …, *n*- 1

2. *i* için şu ikisini arayalım:

   (a) *Tamamlanmış* [*i*] = *false*

   (b) $İhtiyaç_i \leq Çalışan$

   Böyle bir *i* yoksa 4. adıma git

3. $Çalışan = Çalışan + Atanmış_i$
   Tamamlanmış[*i*] = *true*
   İkinci adıma git

4. Eğer her *i* için *Tamamlanmış* [*i*] == true ise sistem güvenli durumdadır.

P_i Prosesi için Kaynak-Atama Algoritması

İstek= P_i prosesi için istek vektörü. Eğer $İstek_i[j] = k$ ise P_i prosesi R_j kaynak türünden k adet örnek ister.

1. Eğer $İstek_i \leq İhtiyaç_i$ ise 2. adıma git. Aksi halde, proses maksimum talebi aştığı için hata mesajı ver
2. Eğer $İstek_i \leq Boş$ ise 3. adıma git. Aksi taktirde yeterli kaynak olmadığı için P_i beklemelidir
3. Durumu aşağıdaki gibi değiştirerek talep edilen kaynakların P_i ye atanmasını sağla:

$$Boş = Boş - Request;$$

$$Atanmış_i = Atanmış_i + İstek_i;$$

$$İhtiyaç_i = İhtiyaç_i - İstek_i;$$

- *Eğer güvenli⇒ kaynaklar Pi ye atanır.*
- *Eğer güvensiz⇒ Pi beklemelidir ve eski kaynak-atama durumuna geri alınır.*

Banker Algoritması Örneği

- P_0 ,..., P_4 olmak üzere 5 adet proses;

  3 kaynak :

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

T_0 anındaki görüntü:

	Atanmış	*Max*	*Boş*
	A B C	*A B C*	*A B C*
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

Örnek (Devam)

- İhtiyaç matrisinin içeriği Max – Atanmış olarak tanımlanmıştır.

	İhtiyaç
	A B C
P_0	7 4 3
P_1	1 2 2
P_2	6 0 0
P_3	0 1 1
P_4	4 3 1

- < P_1, P_3, P_4, P_2, P_0> dizisi güvenlik kriterlerini karşıladığı için sistem güvenli durumdadır.

Örnek: P_1 (1,0,2) kaynağı talep eder

- İstek ≤ Boş ((1,0,2) ≤ (3,3,2))⇒ true olup olmadığını kontrol et.

	Atanmış	*İhtiyaç*	*Boş*
	A B C	*A B C*	*A B C*
P_0	0 1 0	7 4 3	2 3 0
P_1	3 0 2	0 2 0	
P_2	3 0 2	6 0 0	
P_3	2 1 1	0 1 1	
P_4	0 0 2	4 3 1	

- Güvenlik algoritmasının çalıştırılması < P_1, P_3, P_4, P_0, P_2> dizisinin güvenlik kriterlerini karşıladığını gösterir.
- P_4 ün (3,3,0) isteği karşılanabilir mi ?
- P_0 ın (0,2,0) isteği karşılanabilir mi ?

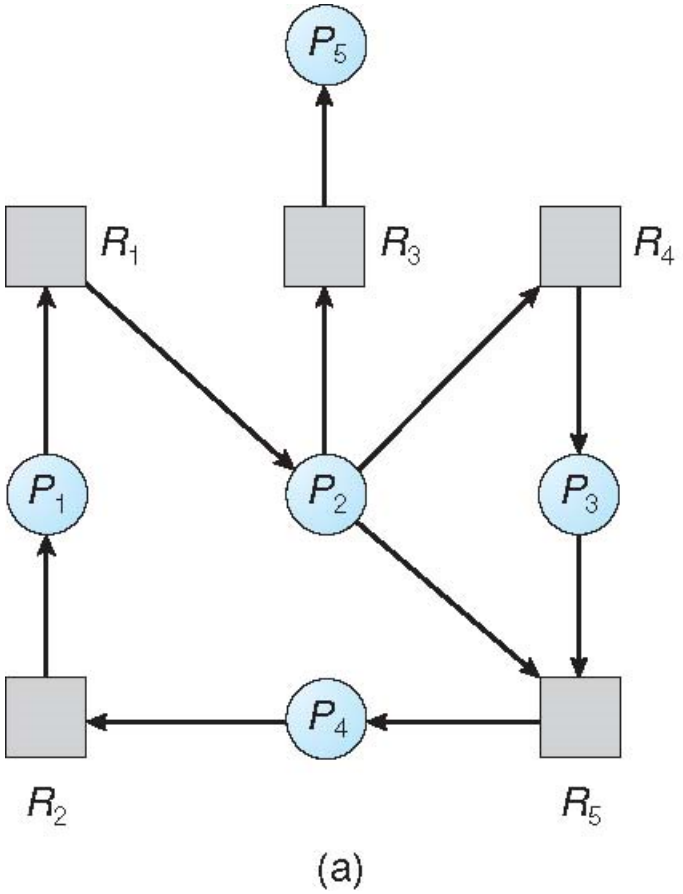
Ölümcül Kilitlenme Tespiti

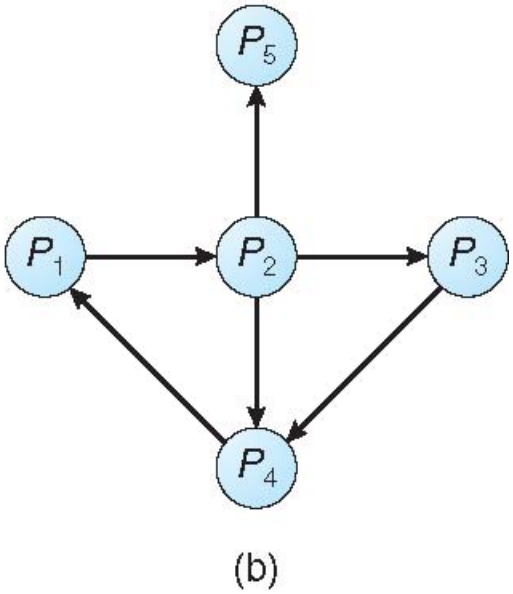
- Sistemin kilitlenme durumuna girmesine izin ver
- Tespit Algoritması
- Kurtarma Şeması

Her Kaynak Türü İçin Tek Örnek

- Bekleme grafı oluştur
  - Düğümler proses
  - $P_i \rightarrow P_j$ eğer P_i P_j yi bekliyorsa
- Periyodik algoritmayı çalıştır.
- Algoritma graf içinde çevrim olup olmadığını arar
- Eğer çevrim varsa ölümcül kilitlenme vardır
- Graf içinde çevrim arayan algoritma n^2 işlem gerektirir
- n graftaki düğümler

Kaynak-Atama ve Bekleme Grafiği





Kaynak-atama grafı

Bekleme grafı

Bir Kaynak Türünden Birkaç Örneği

- **Boş**: *m* uzunluğundaki bir vektör her türdeki mevcut kaynakların sayısını gösterir.
- **Atanmış**: Bir *n* x *m* matrisi, her prosesin o anda sahip olduğu her türden kaynağın sayısını belirtir.
- **İstek**: Bir *n* x *m* matrisi, her prosesin geçerli isteğini gösterir. Eğer *İstek*[*i*][*j*] = *k ise* P_i prosesi ilave k tane Rj tipinden kaynak istiyordur.

Tespit Algoritması

1. *Çalışan* ve *Tamamlanmış* sırasıyla *m* ve *n* uzunluğunda vektörler olsun, başlangıçta:
   (a) *Çalışan* = *boş*
   (b) *i* = 1,2, …, *n için*, eğer $Atanmış_i \neq 0$, ve
   Tamamlanmış[i] = false; aksi halde, *Tamamlanmış*[i] = *true*

2. *i* için şu ikisini arayalım:
   (a) *Tamamlanmış*[*i*] == *false*
   (b) $İstek_i \leq Çalışan$

   Eğer böyle bir *i* yok ise, 4'ünü adıma git

Tespit Algoritması (Devam)

3. *Çalışan* = *Çalışan* + *Atanmış*$_i$
   Tamamlanmış[*i*] = *true*
   2. Adıma git

4. Eğer *i*, $1 \leq i \leq n$ için *Tamamlanmış*[*i*] == false ise sistem kilitlenme durumundadır. Ayrıca, *Tamamlanmış*[*i*] == *false* ise P_i kilitlenmiştir.

Algoritma sistemin ölümcül kilitlenmede olup olmadığı tespit etmek için O(m x n^2) işlem gerektirir

Tespit Algoritması Örneği

- P_0 ,…, P_4 olmak üzere 5 process; 3 kaynak tipi A (7 örnek), *B* (2 örnek), ve *C* (6 örnek)

- T_0'daki anlık görüntüsü :

	Atanmış	*İstek*	*Boş*
	A B C	*A B C*	*A B C*
P_0	0 1 0	0 0 0	0 0 0
P_1	2 0 0	2 0 2	
P_2	3 0 3	0 0 0	
P_3	2 1 1	1 0 0	
P_4	0 0 2	0 0 2	

- <P_0, P_2, P_3, P_1, P_4> dizisi her *i* için *Tamamlanmış* [*i*] = true sonucunu verir.

Örnek (Devam)

- P_2 ek olarak c tipinden bir örnek istiyor.

	İstek
	A B C
P_0	0 0 0
P_1	2 0 2
P_2	0 0 1
P_3	1 0 0
P_4	0 0 2

- Sistemin durumu?
  - P_0 prosesi tarafından tutulan kaynaklar talep edilebilir, ancak diğer prosesler için yetersiz kaynak vardır
  - P_1, P_2, P_3, ve P_4 prosesleri için kilitlenme mevcuttur.

Tespit Algoritması Kullanımı

- «Ne zaman ve ne sıklıkla çağrılmalı?» sorusu aşağıdakilere bağlıdır:
  - Ne sıklıkta kilitlenme meydana gelebilir?
  - Kaç işlemin geri alınması gerekir?
    - Herbir çevrim için bir adet
- Eğer tespit algoritması rasgele olarak çağrılmışsa, kaynak grafında bir çok döngü olabilir ve bu yüzden hangi kilitlenmiş processin kilitlenmeye sebep olduğunu söylememiz mümkün olmaz.

Kilitlenmeden Çıkış: Process İptali

- Kilitlenmiş tüm prosesler iptal edilir
- Kilitlenme döngüsü ortadan kaldırılana kadar prosesler bir bir iptal edilir.
- İptal edilecek prosesi hangi sırayla seçmeliyiz?
  - Prosesin önceliğine göre
  - Prosesin ne kadarı gerçekleşti ve tamamlanması için daha ne kadar süre var?
  - Prosesin kullandığı kaynaklar
  - Prosesin tamamlanması için gerekli kaynaklar
  - Kaç tane prosesi sonlandırmak gerekir?
  - Process etkileşimli mi yoksa toplu iş dosyası (batch) mı?

Kilitlenmeden Çıkış: Kaynak Önceliği

- Bir kurban seçilir – zararı azalt
- Geri alma – güvenli duruma geri dön, bu durum için prosesi yeniden başlat
- Açlık – maliyet faktöründe geri alma sayısını içeren aynı proses her zaman kurban olarak seçilebilir

Bölüm 7 Sonu