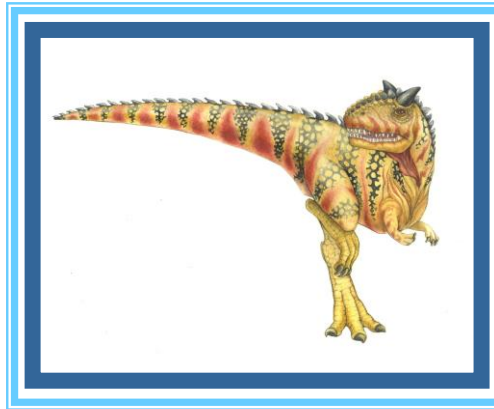
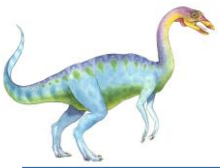


Bölüm 7: Ölümcül Kilitlenme (Deadlocks)





Bölüm 7: Ölümçül Kilitlenme

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





Bölümün Hedefleri

- Eşzamanlı proseslerin görevlerini tamamlamasını engelleyen ölümcül kilitlenmeyi tanımlamak
- Bir bilgisayar sisteminde ölümcül kilitlenmeleri önlemek veya kaçınmak için farklı metotlar sunmak
- Muteks kilitler kullanıldığında kilitlenmenin nasıl ortaya çıkabileceğini gösterilmesi
- Kilitlenmeyi karakterize eden dört gerekli koşulun tanımı
- Kaynak tahsis grafiğinde kilitlenme durumunu belirleme
- Deadlock'ları önlemek için dört farklı yaklaşımın değerlendirilmesi
- Banker algoritması ile deadlock'tan kaçınma
- Deadlock algılama algoritması
- Kilitlenmeden kurtulmak için yaklaşımların değerlendirilmesi





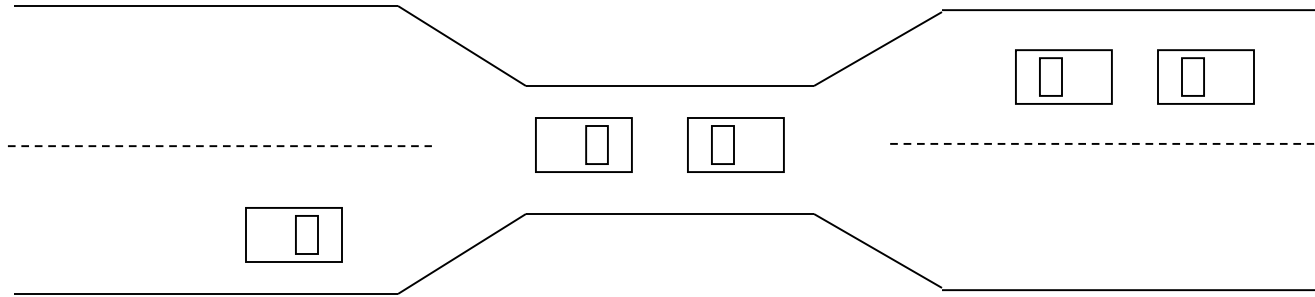
Ölümçül Kilitlenme Problem,

- 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 1 ile başlatır
- | | |
|-----------|----------|
| P_0 | P_1 |
| 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, \dots, 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:
 - **İstek (Request):** Eğer kaynak müsait değilse bekler, müsait olunca kullanabilir
 - **Kullan (Use):** Kaynak kullanılır. Örneğin yazıcı
 - **Serbest bırak (Release):** Kaynak serbest bırakılır





Ölümcül Kilitlenme Karakterizasyonu

Ölümcül kilitlenme aşağıdaki dört durum aynı anda olduğunda ortaya çıkar:

- **Karşılıklı dışlama:** Bir anda sadece bir proses bir kaynağı kullanabilir. Proses kaynağı bırakana kadar diğerleri bekler.
- **Tut ve bekle:** Bir prosesin en az bir kaynak tutması ve şu anda diğer prosesler tarafından tutulmakta olan ek kaynakları edinmesi için beklemesi gerekir.
- **Kesinti yok:** Bir kaynak sadece onu elinde tutan proses tarafından gönüllü olarak serbest kalır, sonra proses görevini tamamlar
- **Döngüsel bekleme:** $\{P_0, P_1, \dots, P_n\}$ 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.





Deadlock Example

```
/* Create and initialize the mutex locks */
pthread_mutex_t first_mutex;
pthread_mutex_t second_mutex;
pthread_mutex_init(&first_mutex, NULL);
pthread_mutex_init(&second_mutex, NULL);

/* thread one runs in this function */
void *do_work_one(void *param)
{
    pthread_mutex_lock(&first_mutex);
    pthread_mutex_lock(&second_mutex);
    /** * Do some work */
    pthread_mutex_unlock(&second_mutex);
    pthread_mutex_unlock(&first_mutex);
    pthread_exit(0);
}
```



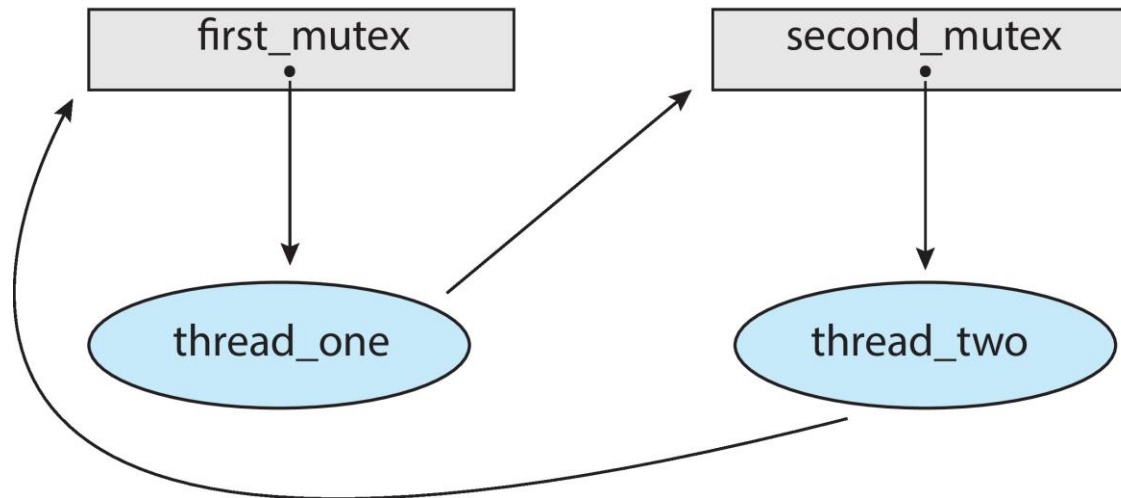


Deadlock Example

```
/* thread two runs in this function */  
void *do_work_two(void *param)  
{  
    pthread_mutex_lock(&second_mutex);  
    pthread_mutex_lock(&first_mutex);  
    /** * Do some work */  
    pthread_mutex_unlock(&first_mutex);  
    pthread_mutex_unlock(&second_mutex);  
    pthread_exit(0); }  

```

- Eğer thread_two iş parçacığı 2. mutex kilidini tutarken thread_one iş parçacığı 1. mutex kilidini elde ederse ölümcül kilitlenme mümkündür.





Kaynak-Atama Grafi

Düğüm kümesini V (*Vertices*) ve kenar kümesini E (*Edge*) ile gösterelim

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





Kaynak-Atama Grafi (Devam)

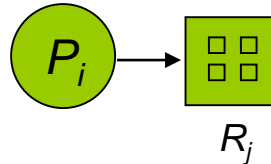
- Proses



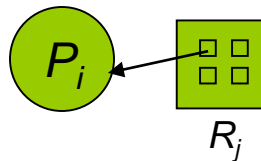
- 4 adet örneğe sahip bir kaynak

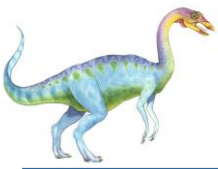


- P_i , R_j 'den bir adet ister



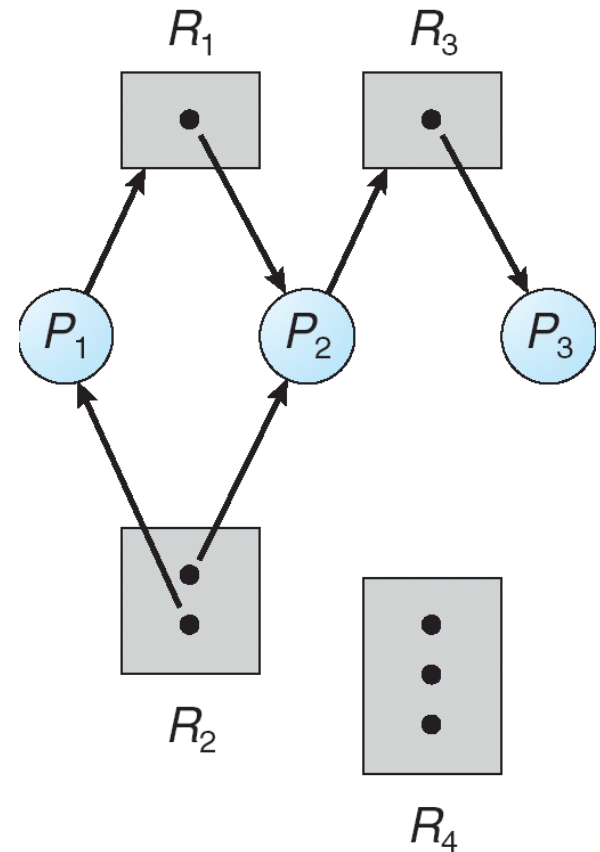
- P_i , R_j 'den bir adetini elinde tutar





Kaynak-Atama Grafi Örneği

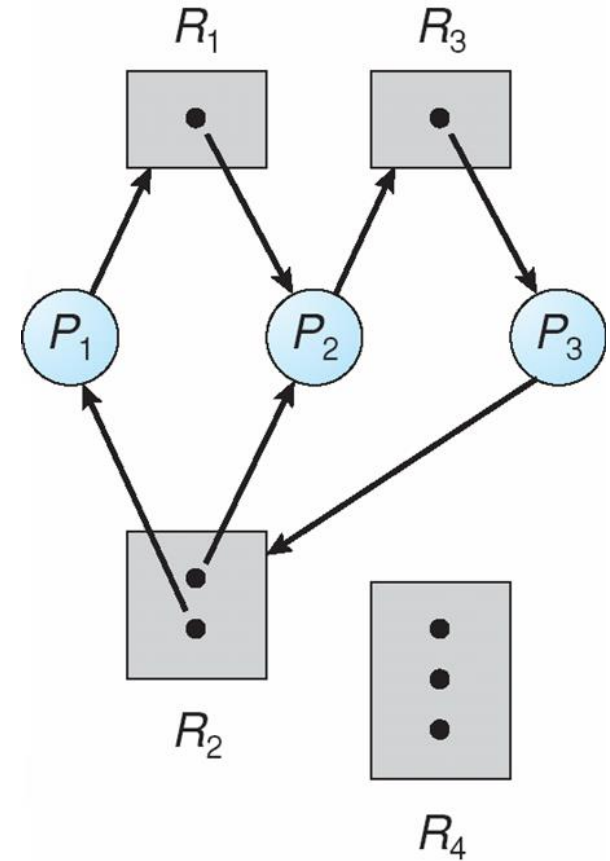
- $P = \{P_1, P_2, P_3\}$
- $R = \{R_1, R_2, R_3, R_4\}$
- $E = \{P_1 \rightarrow R_1, P_2 \rightarrow R_3, R_1 \rightarrow P_2, R_2 \rightarrow P_2, R_2 \rightarrow P_1, R_3 \rightarrow P_3\}$
- P1 bir R2 örneğini tutar ve bir R1 örneğini bekler.
- P2, R1'in bir örneğini, R2'nin bir örneğini tutar ve bir R3 örneğini bekler.
- P3, R3'ün bir örneğini tutuyor





Ölümcül Kilitlenmeli Kaynak-Atama Grafi

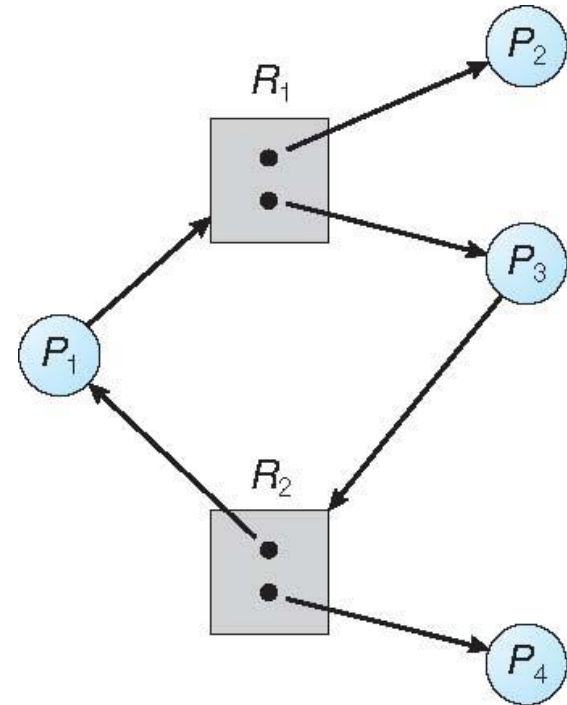
- Bu noktada, sistemde iki minimum çevrim mevcuttur:
- $P1 \rightarrow R1 \rightarrow P2 \rightarrow R3 \rightarrow P3 \rightarrow R2 \rightarrow P1$
- $P2 \rightarrow R3 \rightarrow P3 \rightarrow R2 \rightarrow P2$
- Proses P1, P2 ve P3 kilitlendi





Ölümcül Kilitlenmesiz Ancak Çevrimli Graf

- Bu noktada, sistemde :
- $P_1 \rightarrow R_1 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$
- çevrim mevcuttur ama P_4 , R_2 kaynağından tuttuğu bir örneği bırakabilir ve örnek P_3 'e tahsis edilerek çevrim bozulabilir





Temel Bilgiler

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





Ölümcül Kilitlenme Yönetim Metodları

1. Sistemin **asla** kilitlenme durumuna girmeyeceğini garanti et.
 - Bir protokol kullanarak Ölümcül Kilitlenmeyi Önleme veya Kaçınma sağlanabilir
2. Sistemin bir ölümcül kilitlenme durumuna girmesine izin ver ve daha sonra kurtar.
3. Problemi yok say ve sistemde hiçbir zaman kilitlenme meydana gelmiyor gibi davran; UNIX dahil olmak üzere birçok işletim sistemi tarafından kullanılmıştır.



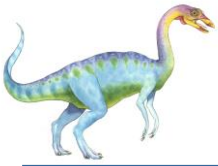


Ölümcül Kilitlenmeyi Önleme

Bir isteğin yapılabileceği yolları kısıtla;

- **Karşılıklı Dışlama** – paylaşılabilir kaynaklar için gerekli değildir (Read-only, yalnızca okunabilir dosyalar gibi); ancak paylaşılabilir 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 herhangi bir kaynağa sahip değilken 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 ve yeni taleplerini almak isteyen proses yeniden başlatılır.

■ Çevrimsel bekleme – Tüm kaynak türlerinin sıralanmasını ve her bir prosesin artan bir sırada kaynakları istemesini şart koş.





Çevrimsel bekleme

- Çevrimsel bekleme koşulunu geçersiz kılmak en yaygın olanıdır.
- Her kaynağı (yani mutex kilitleri) benzersiz bir numara atamanız yeterlidir.
- Kaynaklar sırayla alınmalıdır.
- Eğer:

first_mutex = 1
second_mutex = 5

- **thread_two** için kod aşağıdaki gibi yazılamadı:

```
/* thread.one runs in this function */  
void *do_work_one(void *param)  
{  
    pthread_mutex_lock(&first_mutex);  
    pthread_mutex_lock(&second_mutex);  
    /**  
     * Do some work  
     */  
    pthread_mutex_unlock(&second_mutex);  
    pthread_mutex_unlock(&first_mutex);  
  
    pthread_exit(0);  
}
```

```
/* thread.two runs in this function */  
void *do_work_two(void *param)  
{  
    pthread_mutex_lock(&second_mutex);  
    pthread_mutex_lock(&first_mutex);  
    /**  
     * Do some work  
     */  
    pthread_mutex_unlock(&first_mutex);  
    pthread_mutex_unlock(&second_mutex);  
  
    pthread_exit(0);  
}
```





Ölümcül Kilitlenmeden Kaçınma

Sistemin ilave ön bilgiye sahip olmasını gerektirir

- En Basit ve kullanışlı bir modeli -> her proses 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 sayısı ile tanımlanır





Güvenli Durum

- Bir proses, boşta bir kaynağı talep ettiğinde; bu talebin yerine getirilmesinin sistemi güvenli durumdan çıkarıp çıkarmayacağını işletim sistemi karar vermelidir.
- $\langle P_1, P_2, \dots, P_n \rangle$ sistemdeki sıralanmış tüm prosesleri göstermek üzere herbir P_i için, P_i nin talep ettiği kaynaklar, mevcut boşta kaynaklar + tüm P_j ler tarafından tutulan kaynaklar ile sağlanıyorsa ($j < i$ olmak üzere) **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.





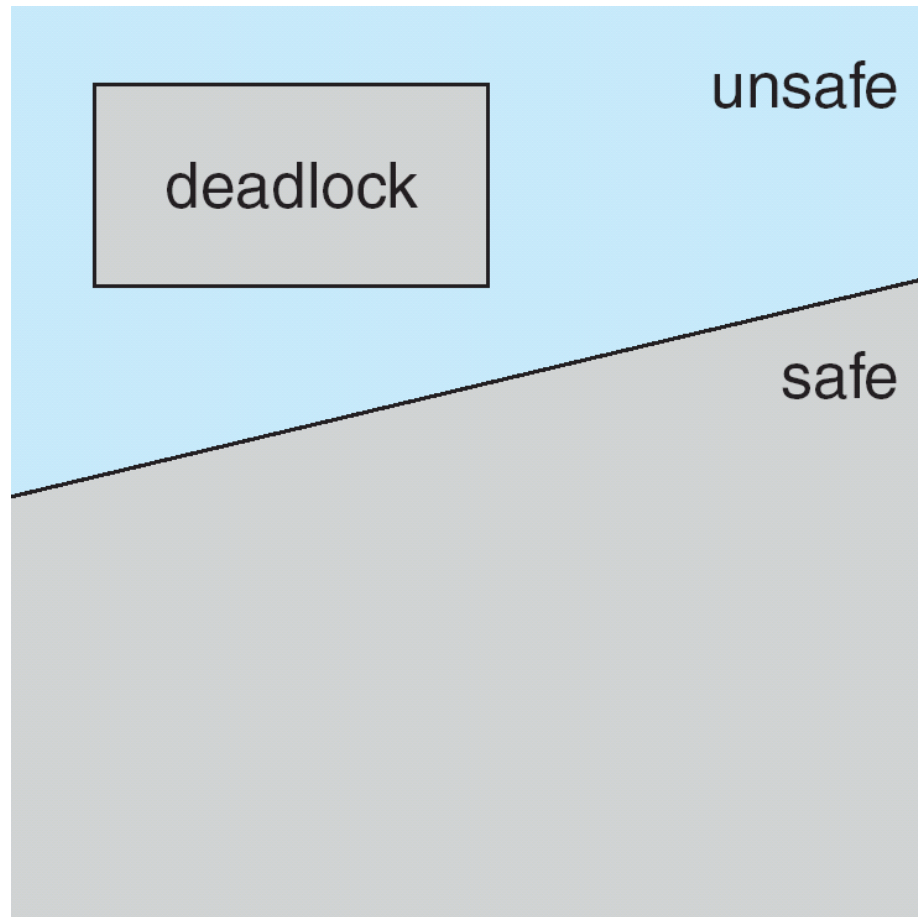
Temel Bilgiler

- Eğer sistem güvenli durumdaysa \Rightarrow kilitlenme yok.
- Eğer sistem güvensiz durumdaysa \Rightarrow kilitlenme olabilir.
- Kaçınma \Rightarrow 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.





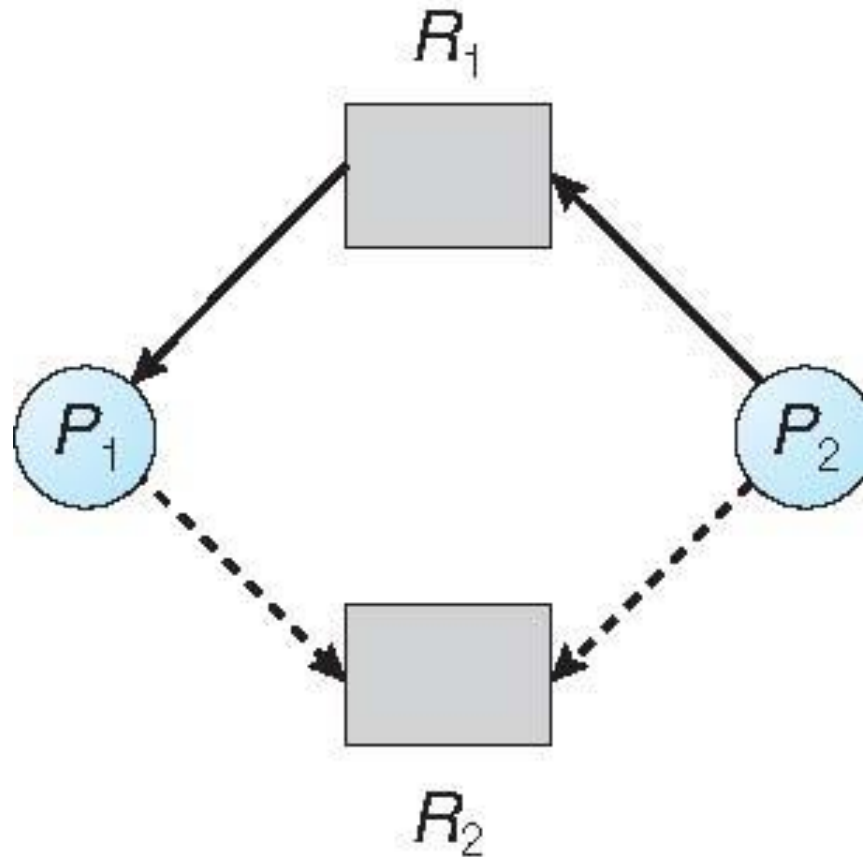
Kaynak-Atama Grafi Şeması

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



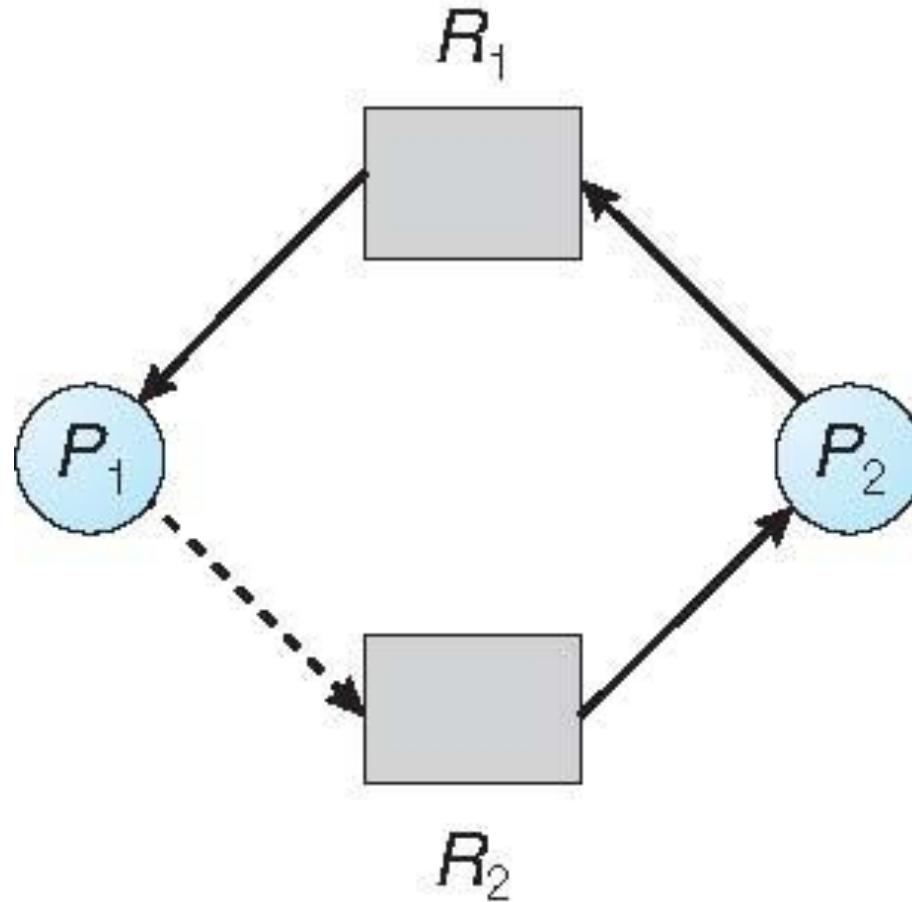


Kaynak-Atama Grafi





Kaynak-Atama Grafında Güvensiz Durum





Kaynak-Atama Grafi 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

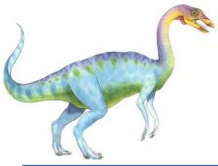




Banker Algoritması

- 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 aldığı anda 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:** $n \times m$ 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:** $n \times m$ boyutunda. Eğer **Atanmış** $[i,j] = k$ ise P_i prosesi k tane R_j örneğini almış durumdadır.
- **İhtiyaç Matrisi:** $n \times m$ boyutunda. Eğer **İhtiyaç** $[i,j] = k$, ise P_i prosesi görevini tamamlamak için ilave k adet R_j örneğine ihtiyaç duymaktadır.

$$\text{İhtiyaç } [i,j] = \text{Max}[i,j] - \text{Atama}[i,j]$$





Güvenli durum (Safety) Algoritması

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

Çalışan = boşta

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

2. İkisini de içeren bir i bulun: :

(a) *Tamamlanmış* $[i] = false$

(b) $İhtiyaç_i \leq Çalışan$

Böyle bir i yoksa 4. adıma git

3. *Çalışan* = *Çalışan* + *Atama_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

$\dot{I}stek_i$ = P_i prosesi için istek vektörü. Eğer $\dot{I}stek_i[j] = k$ ise P_i prosesi R_j kaynak türünden k adet örnek ister.

1. Eğer $\dot{I}stek_i \leq \dot{I}htiyaç_i$ ise 2. adıma git. Aksi halde, proses maksimum talebi aştığı için hata mesajı ver
2. Eğer $\dot{I}stek_i \leq Boş$ ise 3. adıma git. Aksi takdirde yeterli kaynak olmadığı için P_i beklemelidir
3. Durumu aşağıdaki gibi değiştirerek talep edilen kaynakların P_i ye atanmasını sağla:

$$Boş = Boş - Request_i ;$$

$$Atanmış_i = Atanmış_i + \dot{I}stek_i ;$$

$$\dot{I}htiyaç_i = \dot{I}htiyaç_i - \dot{I}stek_i ;$$

- Eğer güvenli \Rightarrow kaynaklar P_i ye atanır.
- Eğer güvensiz \Rightarrow P_i beklemelidir ve eski kaynak-atama durumuna geri alınır.





Banker Algoritması Örneği

- P_0, \dots, P_4 olmak üzere 5 adet proses;

3 kaynak :

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

T_0 anındaki görüntü:

	<u>Atanmış</u>	<u>Max</u>	<u>Boşta</u>
	$A \ B \ C$	$A \ B \ C$	$A \ B \ C$
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	





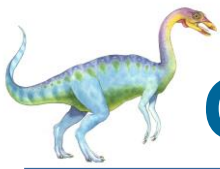
Örnek (Devam)

- İhtiyaç matrisinin içeriği Max – Atanmış olarak tanımlanmıştır.

	<u>İhtiyaç</u>		
	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

- $\langle P_1, P_3, P_4, P_2, P_0 \rangle$ dizisi güvenlik kriterlerini karşıladığı için sistem güvenli durumdadır.





Örnek: P_1 (1,0,2) kaynağı talep eder

- İstek \leq Boş ((1,0,2) \leq (3,3,2)) \Rightarrow true olup olmadığını kontrol et.

	<u>Atanmış</u>	<u>İhtiyaç</u>	<u>Boş</u>
	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 $\langle P_1, P_3, P_4, P_0, P_2 \rangle$ dizisinin güvenlik kriterlerini karşıladığını gösterir.
- P_4 ün (3,3,0) isteği karşılanabilir mi ?
- P_0 ın (0,2,0) isteği karşılanabilir mi ?





Quiz

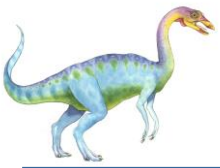
	Allocation			Max		
	X	Y	Z	X	Y	Z
P0	0	0	1	8	4	3
P1	3	2	0	6	2	0
P2	2	1	1	3	3	3

İstek1: P0 (0,0,2) 0 tane X, 0 tane Y ve 2 tane Z

İstek2: P1 (1,1,1) 2 tane X, 0 tane Y ve 0 tane Z

Boşta (3,2,2)





Ölümcül Kilitlenme Tespiti

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

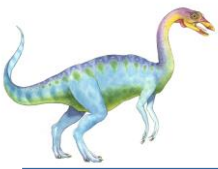




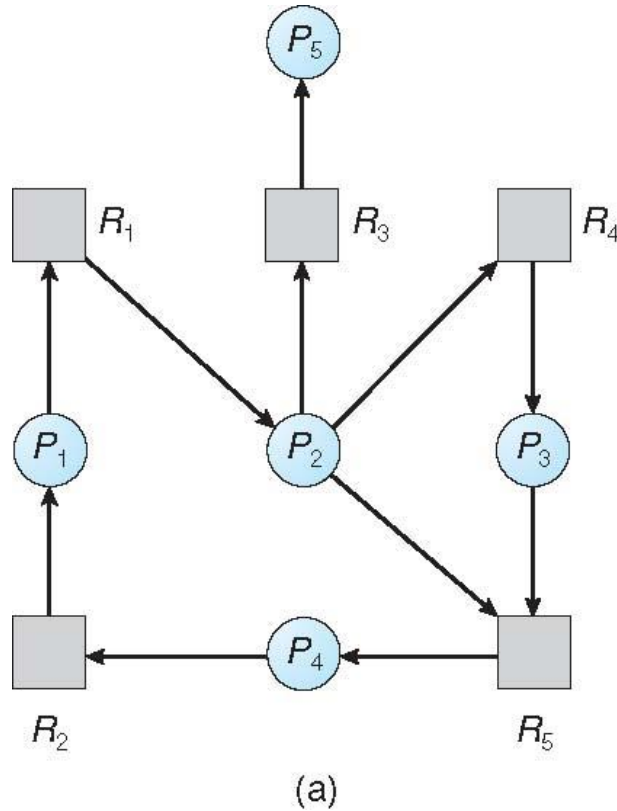
Her Kaynak Türü İçin Tek Örnek

- Bekleme grafi oluştur
 - Döğümler 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 döğümler

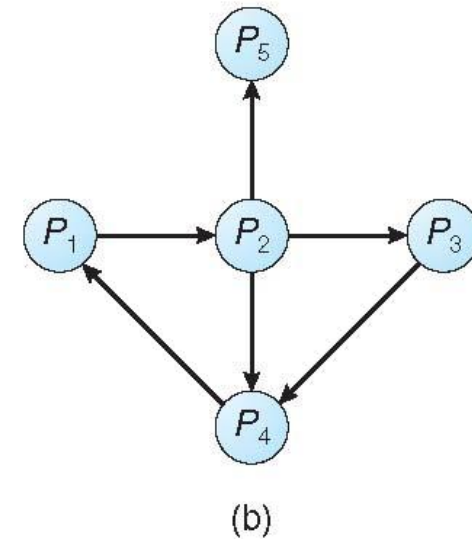




Kaynak-Atama ve Bekleme Grafiği



Kaynak-atama grafi



Bekleme grafi





Bir Kaynak Türünden Birkaç Örneği

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





Tespit Algoritması

1. *Çalışan* ve *Tamamlanmış* sırasıyla m ve n uzunluğunda vektörler olsun, başlangıçta:
 - (a) *Çalışan* = boş
 - (b) $i = 1, 2, \dots, n$ için, eğer $Atanmış_i \neq 0$, ve $Tamamlanmış[i] = \text{false}$; aksi halde, $Tamamlanmış[i] = \text{true}$
2. i için şu ikisini arayalım:
 - (a) $Tamamlanmış[i] == \text{false}$
 - (b) $İstek_i \leq Çalışan$

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





Tespit Algoritması (Devam)

3. $\text{Çalışan} = \text{Çalışan} + \text{Atanmış}_i$
 $\text{Tamamlanmış}[i] = \text{true}$
2. Adıma git
4. Eğer i , $1 \leq i \leq n$ için $\text{Tamamlanmış}[i] == \text{false}$ ise sistem kilitlenme durumundadır. Ayrıca, $\text{Tamamlanmış}[i] == \text{false}$ ise P_i kilitlenmiştir.

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





Tespit Algoritması Örneği

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

- T_0 'daki anlık görüntüsü :

	<u>Atanmış</u>	<u>İstek</u>	<u>Boş</u>
	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	

- $\langle P_0, P_2, P_3, P_1, P_4 \rangle$ dizisi her i için *Tamamlanmış* [i] = true sonucunu verir.





Örnek (Devam)

- P_2 ek olarak c tipinden bir örnek istiyor.

	<u>İstek</u>		
	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

