Kilitlenmeler

Dr. Günay TEMÜR

Bölüm 7 : Kilitlenmeler

- Kilitlenme Problemi
- Sistem Modeli
- Kilitlenme Tarifi
- Kilitlenmeler için Çözüm Yöntemleri
- Kilitlenme Önleme
- Kilitlenmeden Kaçınma
- Kilitlenme Tespiti
- Kilitlenmeden Kurtulma

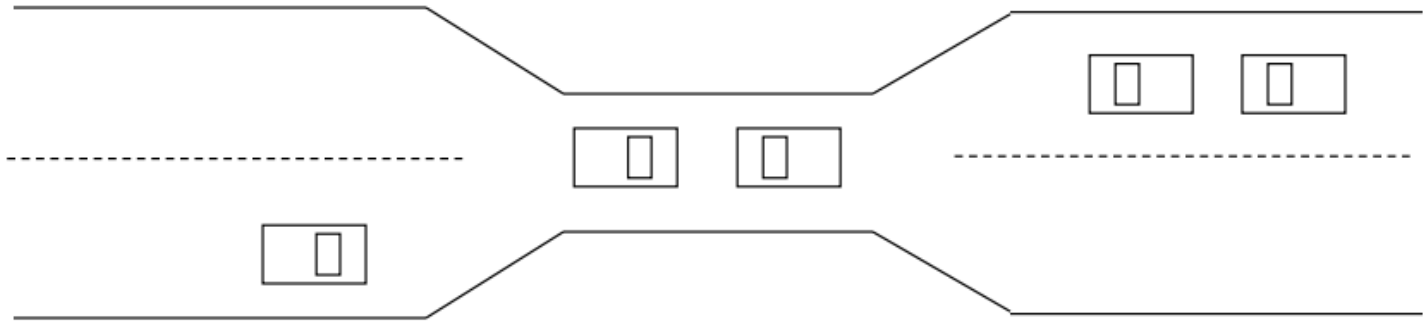
Kilitlenme Problemi

- Her biri bir kaynağı tutan işlemlerden oluşan bir kümede, aynı işlemlerin bu kümedeki bir başka işleme ait kaynağı elde etmek için beklemesi sonucu bloklanmalar
- Örnek
  - Sistem 2 disk sürücüsüne sahip
  - P0 ve P1 işlemlerinin her biri bir diski kullanıyor ve her biri diğer diske ihtiyaç duyuyor

P0	P1
wait(A);	wait(B);
wait(B);	wait(A);

Köprüden Karşıya Geçme Örneği

- Köprü üzerinde trafik tek yöne akabilir
- Köprü bir kaynak gibi görülebilir
- Eğer kilitlenme olursa, araçlardan biri geriye giderse sorun çözülebilir
- Eğer kilitlenme olursa birden fazla aracın geri çekilmesi gerekebilir
- Açlık (starvation) mümkündür
- Not: Pek çok işletim sistemleri kilitlenmelere engel olmaz veya kilitlenmeleri tespit edip çözmeye çalışmaz



Sistem Modeli

- Kaynak tipleri $R_1, R_2, \ldots, R_m$
  - CPU döngüleri (CPU cycles), hafıza alanı, I/O cihazları
- İşlemler bir kaynağı aşağıdaki şekilde kullanırlar:
  - **istek (request)**
  - **kullanım (use)**
  - **iade etme (release)**

Kilitlenme Tarifi

Kilitlenme dört şartın aynı anda sağlanması durumunda ortaya çıkar.

- **Birbirini dışlama (mutual exclusion):** belli bir anda sadece bir işlem, bir kaynağı kullanmalıdır
- **Tutma ve Bekleme (hold and wait):** en az bir kaynağı tutan bir işlem başka işlemler tarafından tutulan ek kaynaklar için beklemelidir
- **Kesinti Yokluğu (no preemption):** bir kaynak ancak o kaynağı tutan işlem, kaynağı isteyerek bırakırsa serbest kalmalıdır
- **Dairesel Bekleme (circular wait):** öyle bir bekleyen işlem kümesi olmalıdır ki $\{P_0, P_1, \ldots, P_n\}$, P_0, P_1 tarafından tutulan bir kaynağı beklemeli, P_1, P_2 tarafından tutulan bir kaynağı beklemeli,…, $P_n–1$, P_n tarafından tutulan bir kaynağı beklemeli ve P_n, P_0tarafından tutulan bir kaynağı beklemelidir.

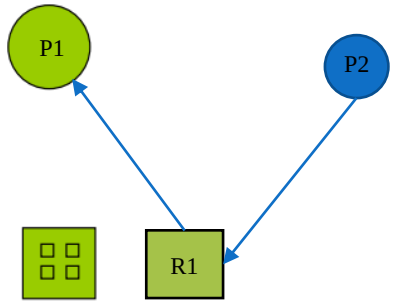
Kaynak-Ayrım Çizgesi (1/2)

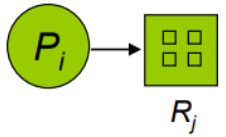
Resource-Allocation Graph V noktalar kümesi ve E kenarlar kümesi

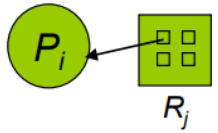
- V iki farklı tipe bölünüyor:
  - P = {P_1, P_2, …, P_n}, sistemdeki tüm işlemlerin kümesi
  - R = {R_1, R_2, …, R_m}, sistemdeki tüm kaynakların kümesi
- İstek kenarı (request edge) – $P_i \rightarrow R_j$ yönlü kenar
- Atama kenarı (assignment edge) – $R_j \rightarrow P_i$ yönlü kenar

Kaynak-Ayrım Çizgesi (2/2)

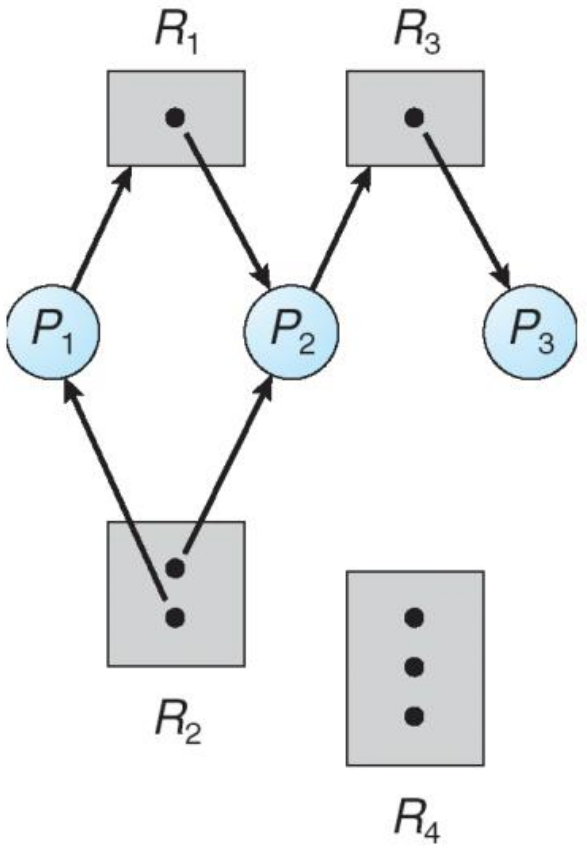
- İşlem
- 4 örneği olan kaynak tipi
- P_i, R_j örneğini istiyor
- P_i, R_j örneğini tutuyor



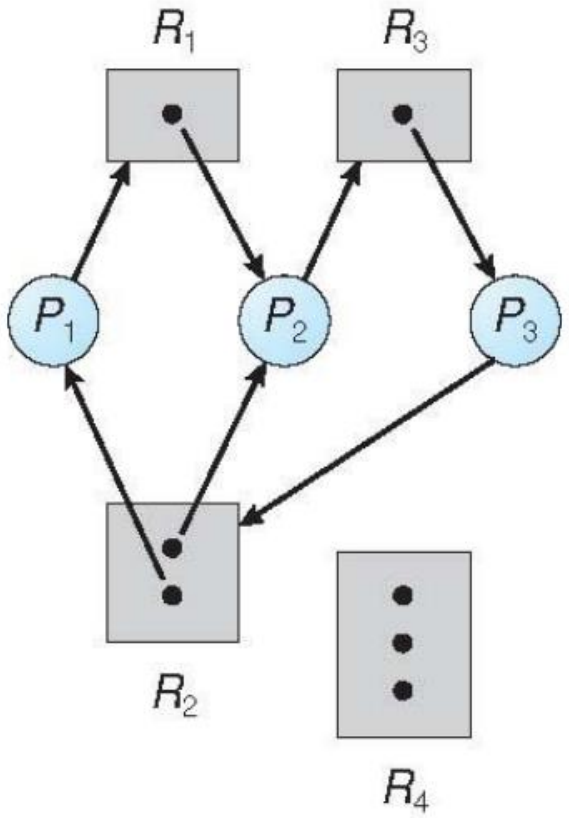




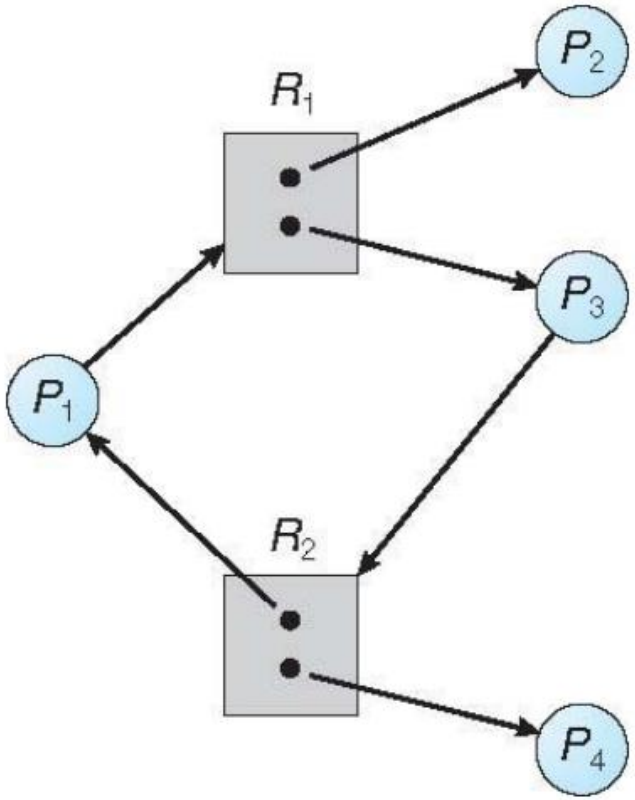
Örnek Kaynak-Ayrım Çizgesi



Kilitlenme İçeren Kaynak-Ayrım Çizgesi



Döngü İçeren Fakat Kilitlenme İçermeyen Çizge



Kaynak-Ayrım Çizgesi Temel Bilgileri

- Eğer çizge döngü içermiyorsa → kilitlenme yok
- Eğer çizge döngü içeriyorsa →
  - Her bir kaynak tipi için bir örnek varsa, kilitlenme var
  - Her bir kaynak için birden fazla örnek varsa, kilitlenme olma olasılığı var

Java Kilitlenme Örneği

```
public static void main(String arg[]) {
   Lock lockX = new ReentrantLock();
   Lock lockY = new ReentrantLock();

   Thread threadA = new Thread(new A(lockX,lockY));
   Thread threadB = new Thread(new B(lockX,lockY));

   threadA.start();
   threadB.start();
```

- Eğer aşağıdaki senaryo gerçekleşirse kilitlenme mümkün:
- threadA -> lockY -> threadB -> lockX -> threadA

Kilitlenmeler İçin Çözüm Yöntemleri

1. **Problemi yok say:** Sistemde kilitlenmelerin hiç bir zaman oluşmayacağını varsay. Pek çok işletim sistemi bu yöntemi kullanır: UNIX ve Java gibi
2. **Kilitlenmeyi Önleme:** Sistemin asla kilitlenme durumuna geçmesine izin verme.
3. **Kilitlenmeyi Çözme:** Sistemin kilitlenmesine izin ver. Ardından kilitlenme durumunu algılayıp kilitlenmeyi çöz

Kilitlenmeyi Önleme 1 - Deadlock Prevention

Daha önce belirtildiği gibi, kilitlenme olması için dört gerekli şart sağlanmalıdır.

Bu şartlardan en az biri sağlanmazsa, kilitlenmenin önüne geçilebilir.

- Birbirini dışlama (mutual exclusion) – paylaşılamaz kaynaklar için şart (örn: yazma işlemi), paylaşılabilir kaynaklar için gerekli değil (örn: okuma işlemi)
- Tutma ve Bekleme (hold and wait) – bir işlem, bir kaynak için istekte bulunduğunda, başka bir kaynağı tutmadığının garantilenmesi gerekir. Örnek bir protokol:
  - İşlemin, ancak sahip olduğu kaynağı iade ettikten sonra yeni kaynak istemesine izin verilebilir
  - Problemler
    - Düşük performanslı kaynak kullanımı
    - Açlığın (starvation) mümkün olması

Kilitlenmeyi Önleme 2

- Kesinti Yokluğu (no preemption) –
  - Eğer bazı kaynaklara sahip bir işlem, direk elde edemeyeceği bir kaynağı isterse, tüm sahip olduğu kaynakları elinden alınır ve bekleme moduna alınır
  - Geri alınan kaynaklar, bu kaynakları bekleyen işleme verilir
  - Kaynakları alınan işlem, ancak eski kaynaklarını ve yeni istediği kaynakları alabileceği zaman yeniden başlatılır
  - Bu protokol, genellikle, durumu kolayca kaydedilip eski haline gelebilecek kaynaklar için kullanılır (yazmaçlar ve hafıza alanı gibi)

Kilitlenmeyi Önleme 3

- Dairesel Bekleme (circular wait) – kaynaklara erişimi tam anlamıyla sıraya koyarak ve işlemlerin kaynakları bu sıraya uygun şekilde elde etmesini sağlayarak engellenebilir

  F(teyp sürücüsü) = 1

  F(disk sürücüsü) = 5

  F(yazıcı) = 12

- Hem yazıcıyı hem de teyp sürücüsünü elde etmek isteyen işlemci, önce teyp sürücüsünü, ardından yazıcıyı istemelidir

Example

```
void transaction(Account from, Account to, double amount)
{
   mutex lock1, lock2;
   lock1 = get_lock(from);
   lock2 = get_lock(to);
   acquire(lock1);
      acquire(lock2);
         withdraw(from, amount);
         deposit(to, amount);
      release(lock2);
   release(lock1);
}
```

Transactions 1 and 2 execute concurrently. Transaction 1 transfers $25 from account A to account B, and Transaction 2 transfers $50 from account B to account A

Kilitlenmeden Kaçınma 1- Deadlock Avoidance

- Önceki slaytlarda anlatılan kilitlenme önleme algoritmaları, kilitlenmenin dört ön şartından en az birinin oluşmasını engelleyerek çalışıyordu
- Bu ise; sistemin ve cihazın kullanım performansının düşmesine neden olmaktadır
- Kilitlenme önlemede alternatif bir yöntem, kaynakların nasıl isteneceğine dair ek önbilgi gerektirir
- İhtiyaç duyulan önbilginin miktarına ve tipine göre farklı algoritmalar bulunmaktadır

Kilitlenmeden Kaçınma 2

- En basit ve en kullanışlı model, işlemlerin her bir tip kaynaktan en fazla kaç tane ihtiyaç duyabileceğini belirtmesini gerektirir
- Bu bilgileri elde eden kilitlenme kaçınma algoritması dinamik olarak kaynak atama durumunu inceler ve dairesel-bekleme (circular-wait) durumunun oluşmasına izin vermez
- Kaynak atama durumu, kullanılabilir ve boştaki kaynak sayısı ile işlemlerin maksimum istekleri ile belirlenir

Güvenli Durum

- Bir işlem boştaki bir kaynağı istediğinde, sistem kaynağın bu işleme verilmesi durumunun sistemi kilitlenmeye neden olmayacak güvenli durumda bırakıp bırakmayacağına karar vermelidir
- Eğer sistem kaynakları tüm işlemlere kilitlenme olmaksızın belirli bir işlem sırasında sağlayabiliyorsa, sistem güvenli durumdadır
- Sistemdeki $<P_1, P_2, \ldots, P_n>$ işlem sırasını ele alalım
- $j < i$ iken, P_i 'nin istediği tüm kaynaklar, boştaki kaynaklar ve tüm P_j işlemlerinin tuttuğu kaynaklar tarafından sağlanıyorsa sistem güvenli durumdadır
- Bu şart sağlanmazsa sistem güvenilmez (unsafe) durumdadır
- Her güvenilmez durum kilitlenmeye neden olmak zorunda değildir, ancak kilitlenmeye neden olabilir

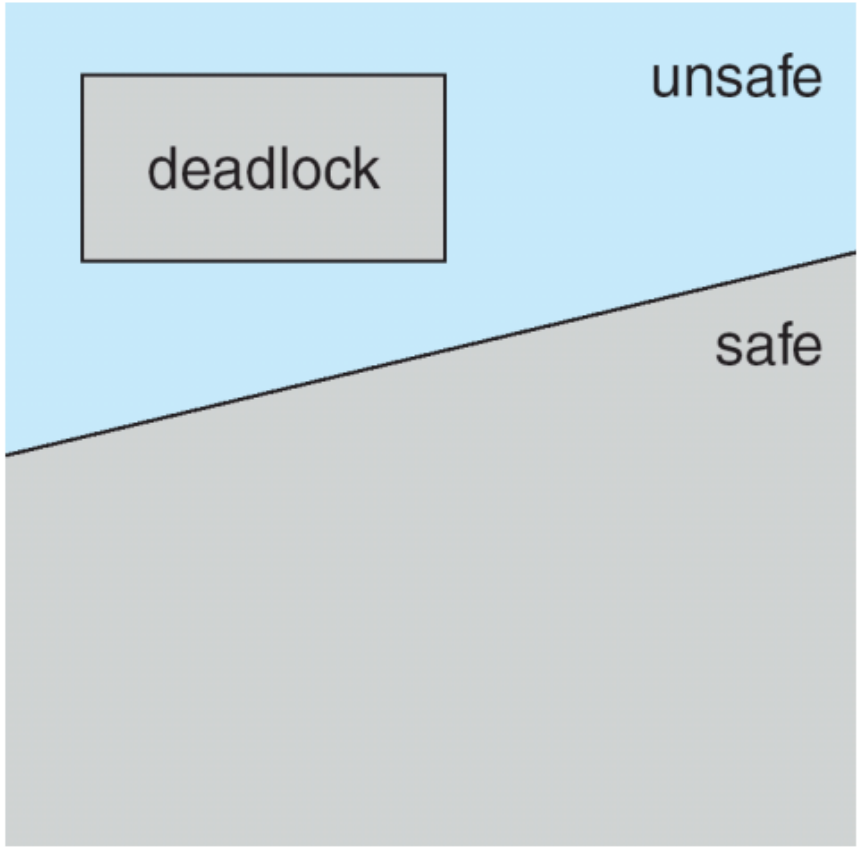
Güvenli Durum

- Genel mantık:
  - Eğer P_i kaynak istekleri o an için karşılanamıyorsa, P_i, P_j işlemlerinin tümünün sonlanmasını bekleyebilir
  - P_j sonlandığında, P_i ihtiyaç duyduğu kaynakları elde edebilir, çalışır ve normal şekilde sonlanır
  - P_i sonlandığında, P_i +1 ihtiyaç duyduğu kaynakları elde edip çalışmaya devam eder ...

Güvenli Durum – Temel Bilgiler

- Eğer sistem güvenli durumdaysa → kilitlenme olmaz
- Eğer sistem güvenilmez durumdaysa → kilitlenme ihtimali vardır
- Kilitlenmeden kaçınma → sistemin hiç bir zaman güvenilmez duruma geçmemesi sağlanarak gerçekleştirilebilir

Güvenilir, Güvenilmez ve Kilitlenme Durumları



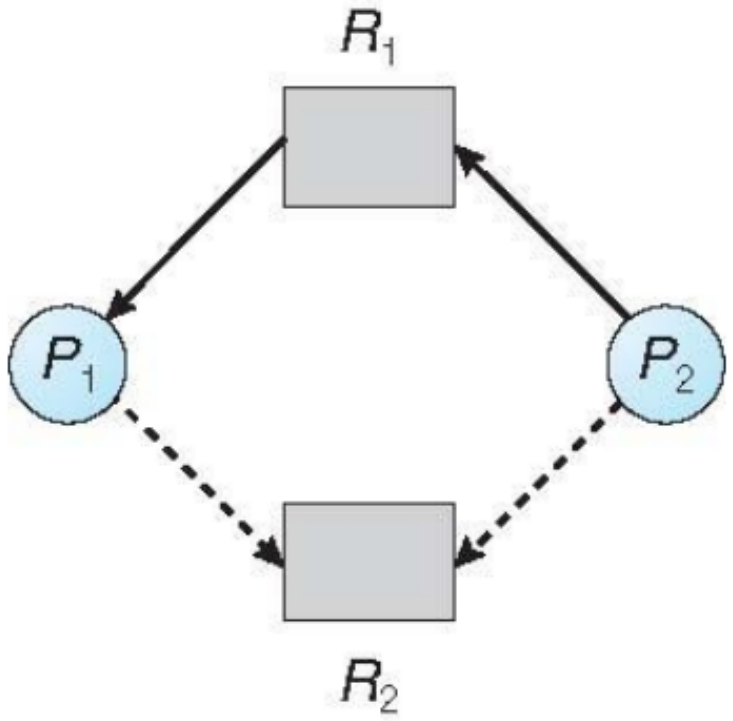
Kilitlenmeden Kaçınma Algoritmaları

- Belirli bir kaynak tipinden bir tane örnek bulunması
  - Kaynak-Ayırım Çizgesi kullanımı
- Belirli bir kaynak tipinden birden fazla örnek bulunması
  - Banker algoritması kullanımı

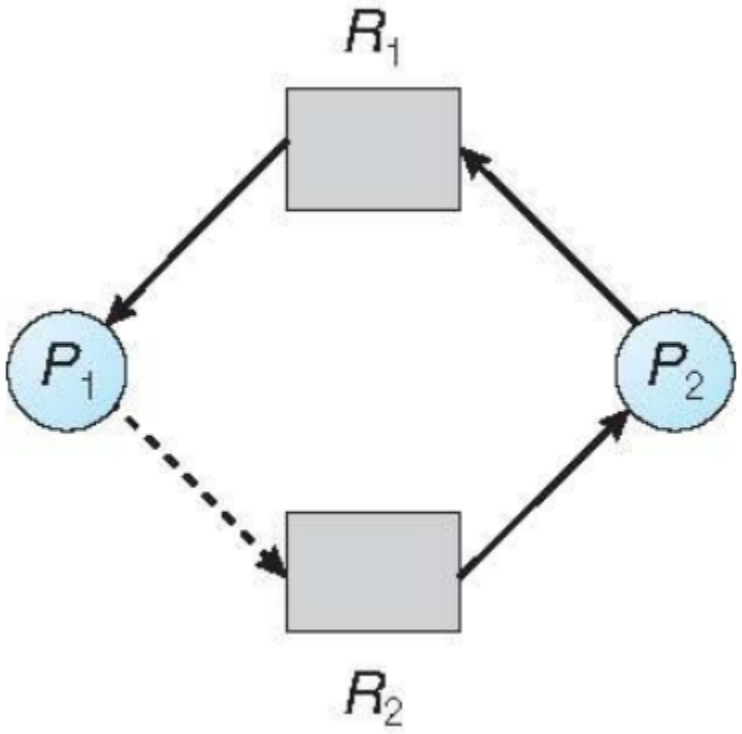
Kaynak-Ayırım Çizgesi Yaklaşımı

- Niyet kenarı (claim edge) Pi → Rj, Pi işleminin ileride bir zamanda Rj kaynağını isteyebileceğini gösteriyor ve çizge üzerinde nokta nokta çizgi (dashed line) şeklinde gösteriliyor
- Niyet kenarı, işlem, kaynağı istediğinde istek kenarına (request edge) dönüştürülüyor
- İstek kenarı, kaynak işleme atandığında atanma kenarına (assignment edge) dönüştürülüyor
- Bir kaynak işlem tarafından iade edildiğinde, atanma kenarı istek kenarına dönüştürülüyor
- İşlemler, sistemdeki kaynaklar için önceden niyetlerini bildirmelidir

Kaynak-Ayrım Çizgesi



Kaynak-Ayrım Çizgesi – Güvenilmez Durum



Kaynak-Ayrım Çizgesi Algoritması

- P_i işleminin R_j kaynağını istediğini varsayalım: istek kenarı
- Eğer istek kenarının atanma kenarına dönüşümü, kaynak-ayırım çizgesinde bir döngünün oluşmasına neden olmuyorsa istek karşılanır
- Aksi halde istek karşılanmaz

Banker Algoritması

- Belirli bir kaynak tipinden birden fazla örnek bulunması
- Her bir işlem maksimum kullanım tahminini, kullanım niyeti olarak bildiriyor
- Bir işlem kaynak talebinde bulunduğunda beklemek zorunda kalabilir
- Bir işlem istediği tüm kaynaklara sahip olduğunda, bu kaynakları sınırlı bir zaman içinde iade etmelidir

Banker Algoritması için Veri Yapıları

n = işlem sayısı m = kaynak tipi sayısı

- Available (uygun): m boyutunda vektör. Eğer Available [j] = k ise, R_j tipinde k örnek kullanıma uygun durumda
- Max (maksimum): n x m matrisi. Eğer Max [i,j] = k ise, P_i işlemi R_j tipinde kaynaklardan en fazla k tanesini isteyebilir
- Allocation (ayırım): n x m matrisi. Eğer Allocation[i,j] = k ise, P_i işlemi şu anda R_j tipindeki kaynaklardan k tanesine sahiptir
- Need (ihtiyaç): n x m matrisi. Eğer Need[i,j] = k ise, P_i işleminin son bulması için R_j kaynak tipinden k tanesine daha ihtiyaç duyabilir

Need [i,j] = Max[i,j] – Allocation [i,j]

P3, P1, P0, P2, P4

Banker Algoritması Örneği

- P0 ... P4; 5 işlem

3 kaynak tipi:

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

T0 anındaki sistem durumu:

	Allocation	Max	Available	
	A B C	A B C	A B C	
P_0	0 1 0	7 5 3	3 3 2	5 3 2 ->p1
P_1	2 0 0	3 2 2		5 4 3 ->p3
P_2	3 0 2	9 0 2		7 4 3 ->p1
P_3	2 1 1	2 2 2		7 5 3 ->p0
P_4	0 0 2	4 3 3		

7 2 5

Banker Algoritması Örneği

- Need matrisinin içeriği (Max – Allocation) olarak tanımlanmıştır:

	Allocation	*Max*	*Available*		Need
	A B C	*A B C*	*A B C*		A B C
P_0	0 1 0	7 5 3	3 3 2	P0	7 4 3
P_1	2 0 0	3 2 2		P1	1 2 2
P_2	3 0 2	9 0 2		P2	6 0 0
P_3	2 1 1	2 2 2		P3	0 1 1
P_4	0 0 2	4 3 3		P4	4 3 1

- Sistem güvenli durumdadır çünkü < P1, P3, P4, P2, P0> sırası güvenlilik kriterini sağlamaktadır

Örnek: P1 İsteği(1,0,2)

- P_1 , (1,0,2) isteğinde bulunmuş olsun
- Request ≤ Available şartının kontrolü: (1,0,2) ≤ (3,3,2) → true
- İstek karşılandıktan sonra, kaynak-atama durumunun güncel hali:

	Allocation	Need	Available
	A B C	A B C	A B C
P0	0 1 0	7 4 3	2 3 0
P1	3 0 2	0 2 0	
P2	3 0 1	6 0 0	
P3	2 1 1	0 1 1	
P4	0 0 2	4 3 1	

- Güvenlik çalıştırıldığında < P1, P3, P4, P0, P2> sırasının güvenlik şartını sağladığı görülür

Örnek: Diğer İstekler

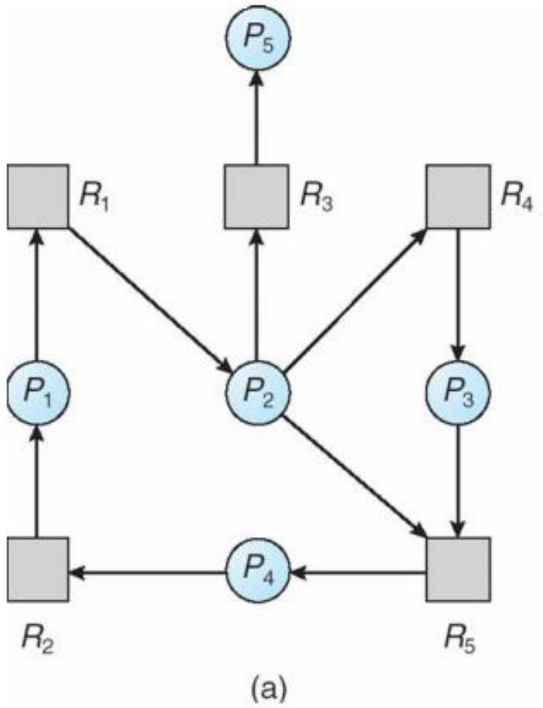
	Allocation	Need	Available
	A B C	A B C	A B C
P0	0 1 0	7 4 3	2 3 0
P1	3 0 2	0 2 0	
P2	3 0 1	6 0 0	
P3	2 1 1	0 1 1	
P4	0 0 2	4 3 1	

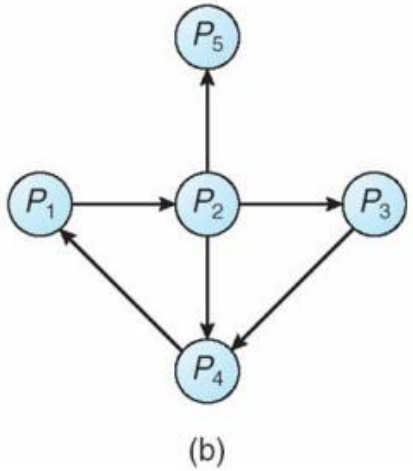
- P4 ‘ün (3,3,0) isteği karşılanabilir mi?
  - Yeterli kaynak yok
- P0‘ın (0,2,0) isteği karşılanabilir mi?
  - Kaynaklar yeterli

Kilitlenme Tespiti ve Kurtarma

- Kilitlenmeyi önleme ve kilitlenmeden kaçınma çözümlerini kullanmayan sistemlerde:
  - Sistemin kilitlenme durumuna girmesine izin verilir
  - Kilitlenme tespit algoritması
  - Kilitlenmeden kurtarma mekanizması

Kaynak-Ayrım Çizgesi ve Bekleme Çizgesi





Kaynak-Ayırım Çizgesi

Karşılık gelen Bekleme Çizgesi

Kaynak Tiplerinden Birden Fazla Örnek Bulunması

- **Available (uygun):** m uzunluğunda vektör. Her bir tip kaynak için kaynaktan kaç tanesinin boşta ve kullanılabilir olduğunu belirtiyor
- **Allocation (ayırım):** n x m boyutlu matris. Her bir tip kaynağın her bir işlem tarafından kadarının kullanıldığını belirtir.
- **Request (istek):** n x m boyutlu matris. Her bir işlemin o anki isteklerini gösterir. Eğer Request [i,j] = k ise, Pi işlemi Rj tipinde kaynaktan k tane daha istiyor demektir

Kilitlenme Tespit Algoritması

Bu algoritma sistemin kilitlenme durumunda olup olmadığını bulmak için O(m x n2) işlem gerektirir

1. Work ve Finish vektörleri sırasıyla m ve n uzunluğunda olsun. . İlk değer ataması:

   (a) Work = Available

   (b) i = 1,2, …, n, eğer $Allocation_i \neq 0$ ise, Finish[i] = false; değilse, Finish[i] = true

2. Aşağıdaki şartları sağlayan i indeksini bul:

   (a) Finish[i] == false

   (b) Requesti ≤ Work

   Eğer bu şartları sağlayan i değerini bulamazsan, 4. adıma git

3. Work = Work + $Allocation_i$

   Finish[i] = true 2. adıma git

4. Eğer herhangi bir i değeri için, 1 ≤ i ≤ n, Finish[i] == false, şartı sağlanıyorsa sistem kilitlenmiştir. Dahası Finish[i] == false ise, Pi işlemi kilitlenmiştir

Tespit Algoritması Örneği

- P_0... P_4; 5 işlem, Üç kaynak tipi: A (7 örnek), B (2 örnek), ve C (6 örnek)
- T_0 anına sistem durumu:

	Allocation	Request	Available
	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> sırası ile çalıştırılırlarsa will tüm *i* değerleri için Finish[*i*] = true olur

Tespit Algoritması Örneği (Devam)

- P_2'nin C tipinde ek bir kaynak daha istediğini düşünelim

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

- Sistemin durumu?
  - P_0'ın bıraktığı kaynaklar geri iade edilir ama diğer işlemlerin isteklerini karşılayacak miktarda kaynak bulunmamaktadır
  - P_1, P_2, P_3, ve P_4 işlemlerinin dahil olduğu bir kilitlenme oluşur

Tespit Algoritması Kullanımı

- Tespit algoritmasının ne zaman ve hangi sıklıkta çalıştırılacağı aşağıdaki faktörlere bağlıdır:
  - Kilitlenme hangi sıklıkta gerçekleşir?
  - Kaç tane işlemin durumu geriye alınmalıdır?
    - Her bir ayrık döngü (disjoint cycle) için bir tane
- Eğer tespit algoritması rastgele çağırılırsa, kaynak çizgesinde pek çok döngü bulunur ve kilitlenmiş pek çok işlem arasında hangi işlemlerin kilitlenmeye neden olduğunu tespit edemeyiz

Kilitlenmeden Kurtarma: İşlemin Sonlandırılması

- Tüm kilitlenen işlemleri sonlandır
- Kilitlenme döngüsü ortadan kalkana kadar, işlemleri teker teker sonlandır
- İşlemleri hangi sırada sonlandırmalıyız?
  - İşlemin önceliğine göre
  - İşlem ne kadar süredir çalışıyor ve sonlanması için ne kadar süreye ihtiyacı var?
  - İşlemin kullandığı kaynaklar?
  - İşlemin tamamlanması için ne kadar kaynağa ihtiyacı var?
  - Kaç tane işlemin sonlandırılması gerekiyor?
  - İşlem interaktif mi yoksa sıralı (batch) işlem mi?

Kilitlenmeden Kurtarma: Kaynak İadesi

- Kurban (victim) seçimi – hangi işleme ait hangi kaynaklar geri alınmalıdır
- Geriye sarma (rollback) – Bir işlemden bir kaynağı geri aldığımızda ne yapacağız?

  Açık bir şekilde işlem çalışmaya normal şekilde çalışmaya devam edemez.
  - İşlemi durdurup yeniden başlatmak
- Açlık (starvation) – bazı işlemler her zaman kurban olarak seçilebilir
  - geriye alınma sayısı seçimin bir parametresi olarak kullanılabilir

BİTTİ