

Sistem modeli

- Multiprogramming ortamlarda çok sayıda process sınırlı kaynağı kullanmak için yarışır.
- Bir process bir kaynağa istek yaptığında, kaynak dolu ise bekleme durumuna geçer.
- **Bekleme durumundaki process, bazı durumlarda hiçbir zaman durumunu değiştiremez.** Buna **deadlock** (kilitlenme) denir.
- İşletim sistemleri genellikle deadlock önleme mekanizmalarını sağlamazlar.
- **Program geliştiricinin deadlock oluşmayacak şekilde program geliştirmesi gereklidir.**

Sistem modeli

- **Bir sistemdeki kaynaklar, CPU cycle, dosyalar, I/O cihazları (yazıcı, DVD sürücü) gibi farklı türdedir.**
- Bir kaynaktan birden fazla varsa (2 CPU, farklı noktalardaki 2 yazıcı), bu kaynakların da birbirinden ayrılması gereklidir.
- Bir process, bir kaynağı kullanmadan önce istek yapar (**request**), kullandıktan sonra da serbest bırakır (**release**).
- Bir process bir kaynağı aşağıdaki sırayla kullanır:
 - **Request:** Process kaynağa istek yapar. **Kaynak kullanılabilir değilse bekler.**
 - **Use:** Process kaynak üzerindeki işlemini gerçekleştirir.
 - **Release:** Process kaynağı serbest bırakır.
- **Cihaz** için **request()** ve **release()**, **dosya** için **open()** ve **close()**, **hafıza** için **allocate()** ve **free()** şeklindedir.

Sistem modeli

- Request ve release işlemleri, **semaforlar** için **wait()** ve **signal()**, **mutex kilitlemesi** için **acquire()** ve **release()** şeklinde tanımlanabilir.
- İşletim sistemleri **kaynakların boş veya atanmış olduğunu bir tablo ile tutarlar.**
- Kaynaklar, **fiziksel** (yazıcı, hafıza alanı, ...) veya **mantıksal** (semafor, mutex lock) olabilir.
- Bir sistemde, 3 tane CD WR sürücüsü olsun. 3 process bu kaynakları kullanırken, **her process diğer CD WR sürücülerden birisine istek yaparsa deadlock oluşur.**
- Bir sistemde, 1 yazıcı ve 1 DVD sürücü olsun. **P0 process'i yazıcıyı, P1 process'i DVD sürücüyü tutarken, P1 yazıcıya ve P0 DVD sürücüye istek yaparsa deadlock oluşur.**
- Multithread uygulama geliştiriciler deadlock olasılığına dikkat etmelidir.

Konular

- Sistem modeli
- **Deadlock tanımı ve özellikleri**
- Deadlock yönetimi için metotlar
- Deadlock önleme
- Deadlock'tan kaçınma
- Deadlock algılama
- Deadlock'tan kurtulma

Deadlock tanımı ve özellikleri

Gerekli şartlar

- Bir **deadlock durumunda**, **process'ler hiçbir zaman sonlanamaz**, **sistem kaynakları atanmış durumdadır** ve **başka işler başlatılamaz**.
- Aşağıdaki 4 şart aynı anda oluştuğunda deadlock ortaya çıkabilir:
 - **Mutual exclusion**: Paylaşımsız kaynak bir process tarafından tutulurken **başka bir process bu kaynağa istek yaparsa**, ikinci **process kaynak boşalıncaya kadar bekler**.
 - **Hold and wait**: Bir process bir kaynağı tutarken, **başka bir kaynağı da bekler durumundadır**. Bu kaynak ise başka bir process tarafından kullanılır durumdadır.
 - **No preemption**: **Kaynaklar önceden boşaltılamaz**. Bir kaynağın boşaltılması için kullanan process'in serbest bırakması gereklidir.
 - **Circular wait**: **{P0, P1, ..., Pn} processleri birbirini beklemektedir**. P0 process'i P1'i, P1 process'i P2'yi, ..., Pn process'i de P0'ı beklemektedir.
- Her durum birbirinden tamamen bağımsız değildir. **Circular wait oluştuğunda, hold and wait durumu vardır**.

7

Deadlock tanımı ve özellikleri

Resource-allocation graph

- Deadlock'lar **bir yönlü graf (directed graph)** kullanılarak **(system resource-allocation graph)** tanımlanabilir.
- Graf üzerinde **düğüm**ler **V (vertices)** ile **kenarlar E (edge)** ile gösterilir.
- **V kümesi** iki kısma ayrılır:
 - $P = \{P_1, P_2, \dots, P_n\}$ aktif process'ler, gösterir.
 - $R = \{R_1, R_2, \dots, R_m\}$ sistemdeki tüm kaynakları gösterir.
- Bir P_i process'inden R_j kaynağına çizilen kenar $P_i \rightarrow R_j$ şeklinde gösterilir.
- $P_i \rightarrow R_j$ kenarı ile P_i process'inin R_j kaynağına **istek yaptığı** ve beklediği ifade edilir.
- $R_j \rightarrow P_i$ kenarı ile de R_j kaynağının P_i process'ine **atandığı** ifade edilir.
- $P_i \rightarrow R_j$ **request edge**, $R_j \rightarrow P_i$ **assignment edge** olarak adlandırılır.

8

Deadlock tanımı ve özellikleri

Resource-allocation graph

- Graf üzerinde her bir **process daire** ile, her bir **kaynak dikdörtgenle** gösterilir.
- Bir kaynaktan birden fazla örnek varsa (birden fazla CD WR) dikdörtgen içerisinde her örnek **ayrı nokta** ile gösterilir.
- Bir process, bir kaynaktan bir örneğe istek yaparsa **request edge** çizilir.
- Bir process'ın yaptığı istek karşılanırsa **assignment edge**'e dönüştürülür.
- Kaynak serbest bırakıldığında ise assignment edge silinerek kaynak serbest bırakılır.

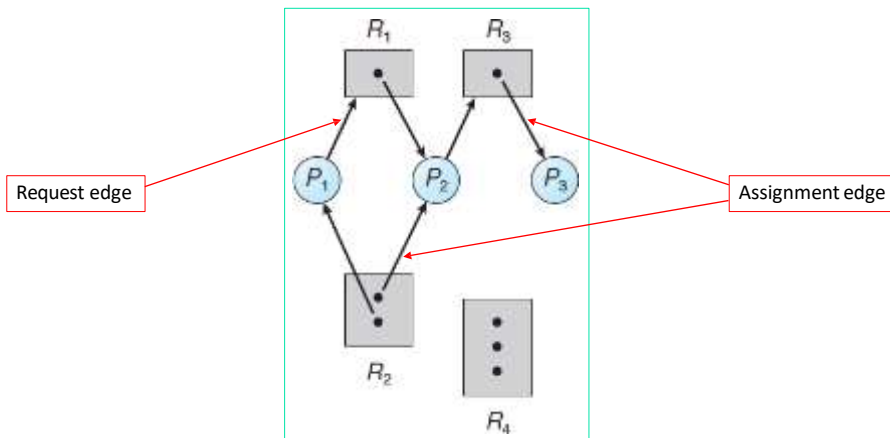
Deadlock tanımı ve özellikleri

Resource-allocation graph

$$P = \{P_1, P_2, P_3\}$$

$$R = \{R_1, R_2, R_3, R_4\}$$

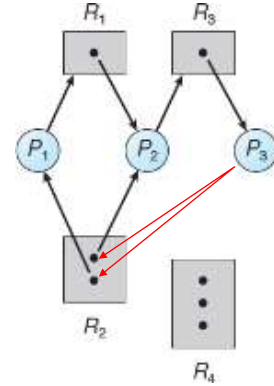
$$E = \{P_1 \rightarrow R_1, P_2 \rightarrow R_3, R_1 \rightarrow P_2, R_2 \rightarrow P_2, R_2 \rightarrow P_1, R_3 \rightarrow P_3\}$$



Deadlock tanımı ve özellikleri

Resource-allocation graph

- Eğer graf üzerinde **döngü yoksa deadlock yoktur, döngü varsa deadlock olabilir.**
- Her kaynaktan sadece 1 örnek varken graf üzerinde döngü varsa, deadlock oluşur.
- Döngü içerisinde yer alan tüm process'ler deadlock durumundadır.
- Kaynaklardan birden fazla olması durumunda, deadlock için döngü oluşması gereklidir ancak yeterli değildir.
- Döngü dışındaki bir process'e atanmış kaynak seçilebilir.

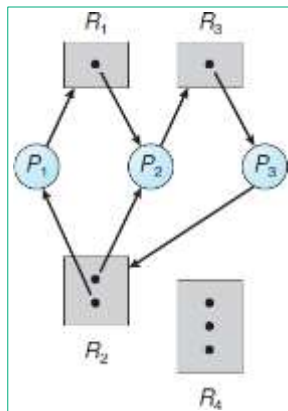


11

Deadlock tanımı ve özellikleri

Resource-allocation graph

- Şekilde P1, P2 ve P3 deadlock durumundadır.



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

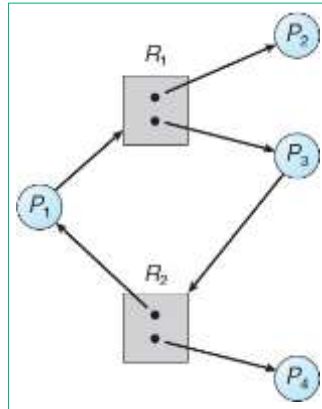
$$P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_2$$

10

Deadlock tanımı ve özellikleri

Resource-allocation graph

- Şekilde döngü vardır, ancak deadlock yoktur.
- P4, R2'nin kopyasını serbest bırakırsa P3 kullanabilir.



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

13

Konular

- Sistem modeli
- Deadlock tanımı ve özellikleri
- **Deadlock yönetimi için metotlar**
- Deadlock önleme
- Deadlock'tan kaçınma
- Deadlock algılama
- Deadlock'tan kurtulma

10

Deadlock yönetimi için metotlar

- Deadlock problemi için 3 farklı yol izlenebilir:
 - **Deadlock'lardan kaçınmak için protokol kullanılabilir.** Sistem hiçbir zaman deadlock durumuna düşmez.
 - Sistemin deadlock durumuna düşmesine izin verilir, **deadlock algılanır ve çözülür.**
 - **Deadlock problemi tamamen gözardı edilir** ve sistemde deadlock hiçbir zaman olmayacak gibi davranılır.
- Üçüncü durum işletim sistemleri tarafından yaygın kullanılır (Linux, Windows).
- **Linux ve Windows işletim sistemleri, deadlock yönetimini uygulama geliştiricilere bırakır.**

15

Deadlock yönetimi için metotlar

- Bir sistemde hiçbir zaman **deadlock olmamasını garanti etmek için, deadlock-prevention** veya **deadlock-avoidance** yöntemleri kullanılabilir.
- **Deadlock prevention, kaynak isteklerini sınırlandırarak** deadlock oluşmasını önler.
- **Deadlock avoidance, bir kaynağa istek yapan process'in yanı sıra, hangi zaman aralığında kullanacağını da bilmek ister.**
- Deadlock avoidance, kaynak isteği yapan process'in beklemesine veya kaynağın atanmasına karar verebilir.
- **Bir sistem, deadlock prevention veya deadlock avoidance yöntemlerini kullanmazsa deadlock oluşabilir.**
- Bu sistemler, **deadlock olup olmadığını kontrol eden bir algoritma ve deadlock oluştuğunda çözümünü sağlayan bir algoritma** sağlamalıdır.

16

Deadlock yönetimi için metotlar

- Bir sistemde **deadlock algılama ve çözme algoritması yoksa**, aynı kaynağa istek yapan ve kilitlenen process sayısı artmaya başlar, bir süre sonra sistem çalışamaz hale gelir ve **en sonunda manuel olarak sistemin restart yapılması gerekir**.
- Deadlock algılama ve çözümü yöntemleri çoğu işletim sisteminde kullanılmaz.
- Bazı sistemler, başka durumlar için kullandığı yöntemleri deadlock yönetiminde de kullanırlar.
- Bazı sistemlerde, deadlock durumu yoktur. Bunun yerine **process'in donmuş durumu (frozen state) vardır**.
- Bu durumda **process işletim sistemine geri dönemez**.

17

Konular

- Sistem modeli
- Deadlock tanımı ve özellikleri
- Deadlock yönetimi için metotlar
- **Deadlock önleme**
- Deadlock'tan kaçınma
- Deadlock algılama
- Deadlock'tan kurtulma

18

Deadlock önleme

Mutual exclusion

- Deadlock oluşması için 4 durumun da (**mutual exclusion, hold and wait, nopreemption, circular wait**) gerçekleşmesi gereklidir.
- Bu şartların birisi engellenirse deadlock önlenmiş olur.
- **Mutual exclusion'da en azından bir kaynak paylaşılamaz (eş zamanlı erişilemez) durumdadır.**
- Paylaşılabilir kaynaklar deadlock oluşturmaz.
- **Read-only dosyalar** paylaşılabilir kaynaklardır ve **deadlock oluşturmaz.**
- **Paylaşılamaz kaynaklara birden fazla process tarafından eş zamanlı erişim yapılamaz** (Semafor veya mutex lock ile engellenir).

19

Deadlock önleme

Hold and wait

- Bir sistemde **hold and wait** durumunun oluşmaması için, bir process bir kaynağa istek yaptığında **başka bir kaynağı tutmaması gerekir.**
- Bir protokol kullanılarak, **bir process çalışmaya başlamadan önce ihtiyaç duyacağı tüm kaynaklar kendisine atanabilir.**
- Başka bir protokolde ise, **eğer bir process kaynak kullanmıyorsa yeni kaynak için istek yapabilir.**
- Bir process, DVD sürücüdeki dosyayı diske kaydetsin, dosyayı sıralasın, ardından yazıcıdan çıktı alsın.
- Birinci protokolde, tüm çalışma süresince DVD sürücü, dosya ve yazıcı process'e atanır (**verimsiz kullanım**).
- İkinci protokolde, kaynaklar ihtiyaç duyduğunda sırayla process'e atanır (**Bir process sık istek yapılan bir kaynağı süresiz bekleyebilir.**).

20

Deadlock önleme

No preemption

- Bir kaynak bir process tarafından tutuluyorsa, **kaynak process tarafından serbest bırakılmadan önce boşaltılamaz.**
- Bir protokol kullanılarak, bir process bir kaynak isteğinde bulunursa ve **bekleme durumuna geçerse, tutmakta olduğu tüm kaynakları serbest bırakır.**
- Başka bir protokol kullanılarak,
 - Bir process bir kaynağa istek yaptığında kullanılabilir olduğu kontrol edilir. Uygunsa tahsis edilir.
 - İstek yapılan kaynak başka process tarafından tutuluyorsa, tutan process'in başka bir kaynağı bekleyip beklemediği kontrol edilir. **Başka bir kaynağı bekliyorsa**, istek yapılan kaynak boşaltılıp (preempt) yeni istek yapan process'e atanır.
 - İstek yapılan kaynak uygun değilse ve kullanan process başka bir kaynağı beklemiyorsa, istek yapan process bekletilir.

23

Deadlock önleme

Circular wait

- Circular wait durumunun önlenmesi için **tüm kaynaklar sıralanır ve bir process kaynak isteğini artan sırada yapabilir.**

$$\begin{aligned}F(\text{tape drive}) &= 1 \\F(\text{disk drive}) &= 5 \\F(\text{printer}) &= 12\end{aligned}$$

- Bir process, **başlangıçta bir kaynağı isteyebilir.** Ardından **yapacağı istekler artan sırada olmak zorundadır.**
- Bir process, R_i kaynağından sonra R_j kaynağını isteyebilir ($F(R_i) < F(R_j)$).
- Eğer bir process **tape drive ile yazıcıyı aynı anda kullanmak isterse, önce tape drive atanır ardından yazıcı atanır.**

23

Konular

- Sistem modeli
- Deadlock tanımı ve özellikleri
- Deadlock yönetimi için metotlar
- Deadlock önleme
- **Deadlock'tan kaçınma**
- Deadlock algılama
- Deadlock'tan kurtulma

23

Deadlock'tan kaçınma

- **Deadlock önleme algoritmaları ile kaynak erişimi sınırlanır.**
- **Deadlock oluşmasını sağlayan durumlardan en az bir tanesi engellenir.**
- Bu durumda, **verimsiz kullanım** ve **düşük throughput** ortaya çıkar.
- **Deadlock'tan kaçınma yöntemlerinde, deadlock oluşumunu engellemek için kaynakların nasıl istendiğinin bilinmesi gerekir:**
 - **P process'i önce tape sürücü ardından yazıcı istemektedir ve ikisini birlikte serbest bırakmaktadır.**
 - **Q process'i ise önce yazıcı ardından tape sürücü istemektedir.**
- **Sistem, ileride deadlock oluşmayacak şekilde kaynakları planlayarak tahsis eder.**
- Basit bir yöntemde, **tüm process'ler ihtiyaç duydukları kaynakları başlangıçta bildirirler**, sistem **circular wait oluşmayacak şekilde kaynakları tahsis eder.**

24

Deadlock'tan kaçınma

Safe state

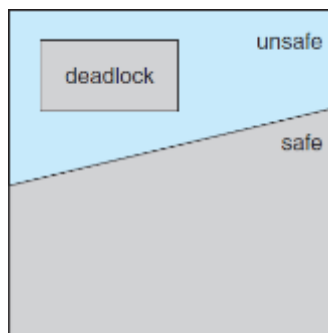
- Eğer bir **sistem**, kaynakları process'lere belirli bir sırada (**safe sequence**) **maksimum ihtiyaçları kadar** atayabiliyorsa ve **deadlock oluşmuyorsa** bu durum **safe state** olarak adlandırılır.
- Eğer bir sistemde **safe sequence** varsa sistem safe state durumundadır.
- $\langle P_1, P_2, \dots, P_n \rangle$ sırası mevcut atama durumu için **safe sequence**'dir, eğer aşağıdaki durumlar sağlanırsa:
 - P_j process'inin tüm kaynak istekleri mevcut **boş kaynaklarla** ve tüm P_i 'ler ($i < j$) tarafından tutulan **dolu kaynaklarla** karşılanır (**tüm P_i 'lerin sonlanması gerekir.**).
 - Eğer P_j 'nin istediği kaynaklar kullanılabilir değilse, tüm P_i 'ler sonlanıp kaynakları boşaltıncaya kadar bekler.
 - P_j kaynak kullanımını **sonlandırdığında**, P_{j+1} process'i kaynağı alıp kullanabilir.

25

Deadlock'tan kaçınma

Safe state

- Bir **safe sequence yoksa** sistem **unsafe durumundadır** (**deadlock oluşabilir**).
- Safe, unsafe ve deadlock state şekilde görülmektedir.



26

Deadlock'tan kaçınma

Safe state - örnek

- Bir sistemde **12 tape sürücü** ve **3 process** (P0, P1 ve P2) olsun.
- P0 process'i maksimum **10 tane**, P1 process'i **4 tane** ve P2 process'i **9 tane** tape sürücüyü ihtiyaç duymaktadır.
- t0 anında, P0 process'i **5 tane**, P1 process'i **2 tane** ve P2 process'i **2 tane** tape sürücü kullanıyor.
- t0 anında, **3 tane tape sürücü boşadır.**

| | Maximum Needs | Current Needs |
|----------------|---------------|---------------|
| P ₀ | 10 | 5 |
| P ₁ | 4 | 2 |
| P ₂ | 9 | 2 |

- t0 anında <P1, P0, P2> **safe sequence**'inde sistem safe durumundadır.

27

Deadlock'tan kaçınma

Safe state - örnek

- <P1, P0, P2> safe sequence için, t1 anında P2 process'i 1 tane kaynak daha alsın (**Safe sequence bozuldu!!!**). **Boş kaynak sayısı 2'ye düşer.**

| | Maximum Needs | Current Needs |
|----------------|---------------|---------------|
| P ₀ | 10 | 5 |
| P ₁ | 4 | 2 |
| P ₂ | 9 | 2 |

3 olur.

- Bu durumda sadece P1 process'i tüm kaynaklarını alabilir (2 tane daha).
- P1 tüm kaynaklarını boşaltsa toplam 4 kaynak boşta olur.
- P0 ve P2 aynı anda tüm kaynakları kullanmak isterse deadlock oluşur.
- P0 process'i 5 kaynak ister, P2 process'i ise 6 kaynak ister (**4 kaynak boş**).
- P2'ye yeni kaynak ataması yapılmadan önce, kendisinden **önceki processlerin tüm kaynaklarını serbest bırakması beklenmelidir.**

28

Deadlock'tan kaçınma

Resource-allocation-graph algorithm

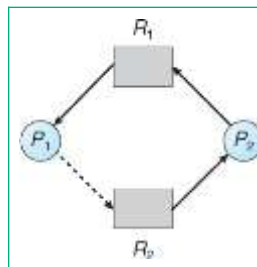
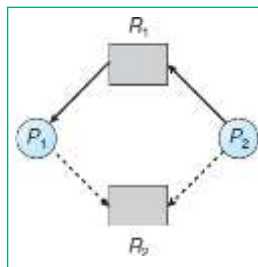
- Her kaynaktan bir örnek olan sistemlerde resource-allocation graf değiştirilerek kullanılabilir.
- Resource-allocation grafına yeni bir kenar eklenir (claim edge).
- $P_i \rightarrow R_j$ claim edge, P_i process'inin R_j kaynağını ileriki zamanda isteyebileceğini gösterir.
- Claim edge graf üzerinde noktalı çizgiyle gösterilir.
- P_i process'i R_j kaynağını istediğinde ise request edge yapılır.
- R_j kaynağı P_i process'i tarafından serbest bırakıldığında, assignment edge claim edge'e dönüştürülür.
- P_i process'i çalışmaya başlamadan önce tüm claim edge'leri graf üzerinde görüntülenir.

25

Deadlock'tan kaçınma

Resource-allocation-graph algorithm

- P_i process'i R_j kaynağını istediğinde claim edge request edge'e dönüşür.
- Assignment edge yapılabilmesi için resource-allocation graf üzerinde döngü oluşmaması gereklidir.
- Şekilde P_2 process'i, R_2 'yi istediğinde boş olmasına rağmen atanmaz (Döngü oluştuğundan sistem unsafe durumuna geçer).
- Daha sonra P_1 process'i R_2 'yi istediğinde deadlock oluşur.



30

Deadlock'tan kaçınma

Banker algoritması

- Resource-allocation algoritması **aynı kaynaktan birden fazla olan sistemlerde uygulanamaz.**
- Aynı kaynaktan birden fazla olan sistemlerde **banker algoritması (banker's algorithm) kullanılabilir.**
- Banka sisteminde, bir banka parasını tüm müşterilerinin ihtiyacını karşılayamayacak seviyeye hiçbir zaman düşürmez.
- **Bir process sisteme geldiğinde tüm kaynaklar için maksimum ihtiyaçlarını bildirir.**
- **Bu kaynak miktarı sistemin tüm kaynaklarından fazla olamaz !!!**
- **Bir process bir grup kaynak istediğinde, sistem bu atamanın yapılması halinde safe state'in korunup korunmadığına bakar.**

33

Deadlock'tan kaçınma

Banker algoritması

- Banker algoritması aşağıdaki veri yapılarını kullanır:
 - **Available:** Bir vektördür. Sistemde boştaki tüm kaynakların sayısını tutar. **Available(j) = k** ise, sistemdeki **R_j** kaynağından **k adet boştedir.**
 - **Max:** $n * m$ matristir. Her process'in maksimum kaynak talebini tutar. **Max[i][j] = k** ise **P_i process'i R_j kaynağından en fazla k tane isteyebilir.**
 - **Allocation:** $n * m$ matristir. Her process'in kullanmakta olduğu kaynak miktarını tutar. **Allocation[i][j] = k** ise, **P_i process'i R_j kaynağından k adet kullanmaktadır.**
 - **Need:** $n * m$ matristir. Her process'in kalan ihtiyaç miktarını tutar. **Need[i][j] = k** ise **P_i process'i R_j kaynağından k adet daha kullanabilir.**
 $Need[i][j] = Max[i][j] - Allocation[i][j]$ olur.
- Veri yapısındaki veri boyutu (satır) ve değerleri dinamik olarak değişebilir (**çalışan process sayısı değişebilir**).

33

Deadlock'tan kaçınma

Banker algoritması - örnek

- Sistemde 5 process vardır. A kaynağından 10, B kaynağından 5 ve C kaynağından 7 tane var (10, 5, 7). T0 anında sistem görüntüsü aşağıdadır.

| | 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 | P ₀ | 7 | 4 | 3 |
| P ₁ | 2 | 0 | 0 | 3 | 2 | 2 | | | | P ₁ | 1 | 2 | 2 |
| P ₂ | 3 | 0 | 2 | 9 | 0 | 2 | | | | P ₂ | 6 | 0 | 0 |
| P ₃ | 2 | 1 | 1 | 2 | 2 | 2 | | | | P ₃ | 0 | 1 | 1 |
| P ₄ | 0 | 0 | 2 | 4 | 3 | 3 | | | | P ₄ | 4 | 3 | 1 |

- Yukarıdaki durumda <P₁, P₃, P₄, P₂, P₀> safe sequence'dir.
- P₁ ihtiyacının tamamını alırsa, boşta (2, 1, 0) kalır. Tümünü bırakırsa (5, 3, 2) boşta'dır. Ardından P₃ tümünü alabilir!!!

33

Deadlock'tan kaçınma

Banker algoritması - örnek

- P₁ = (1, 0, 2) isteğinde bulunursa, öncelikle boşta olanların sağlayıp sağlamadığına bakılır ((1, 0, 2) <= (3, 3, 2)). Boşta kalan (2, 3, 0).
- Kaynak ataması yapıldıktan sonraki durum aşağıdadır.

| | Allocation | | | Need | | | Available | | |
|----------------|------------|---|---|------|---|---|-----------|---|---|
| | 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 | | | |

- Yeni durum için <P₁, P₃, P₄, P₀, P₂> safe sequence'dir.
- Eğer, P₄ = (3, 3, 0) isterse yeterli kaynak olmadığından atanamaz.
- P₀ = (0, 2, 0) atanırsa (P₀ need (7, 2, 3) kalır), boş (2, 1, 0) ve sistem unsafe durumuna geçer.

34

Konular

- Sistem modeli
- Deadlock tanımı ve özellikleri
- Deadlock yönetimi için metotlar
- Deadlock önleme
- Deadlock'tan kaçınma
- **Deadlock algılama**
- Deadlock'tan kurtulma

35

Deadlock algılama

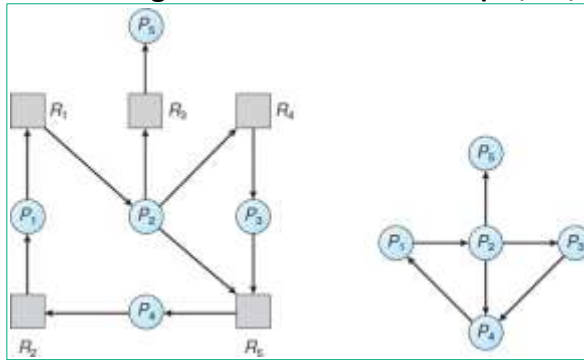
- Bir sistem **deadlock önleme veya deadlock'tan kaçınma algoritmasına sahip değilse**, deadlock oluşabilir.
- Bu tür bir sistem, deadlock probleminin çözümü için aşağıdakileri sağlayabilir:
 - Sistemin deadlock durumunda olup olmadığına karar verecek bir algoritma
 - Deadlock durumunun çözülmesi için bir algoritma
- **Deadlock algılama algoritmaları**, her **kaynaktan bir tane olması** veya her **kaynaktan birden fazla olması** durumuna göre **farklılık gösterir**.

36

Deadlock algılama

Her kaynaktan bir örnek olması

- Resource-allocation grafi kullanılarak **wait-for** grafi oluşturulabilir.
- Wait-for grafında, **kaynak düğümleri kaldırılarak sadece process düğümleri bırakılır.**
- Wait-for grafında **döngü varsa deadlock vardır (P1, P2, P3, P4).**



37

Deadlock algılama

Her kaynaktan birden fazla örnek olması

- Bu tür sistemlerde birden fazla veri yapısı kullanılır:
 - **Available:** Her kaynak türü için boştaki kaynak sayısını tutar.
 - **Allocation:** $n * m$ matristir. Her process için mevcut atanmış kaynak sayısını tutar.
 - **Request:** $n * m$ matristir. Her process'in mevcut istek sayısını tutar.
Request[i][j] = k ise **Pi process'i Rj kaynağından k tane daha istemektedir.**

38

Deadlock algılama

Her kaynaktan birden fazla örnek olması - örnek

- Sistemde 5 process, A = 7, B = 2 ve C = 6 kaynak vardır.
- T0 anındaki kaynak atama durumu aşağıdadır.

| | <u>Allocation</u> | <u>Request</u> | <u>Available</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 | |

- <P₀, P₂, P₃, P₁, P₄> safe sequence olsun.
- P₂ = (0, 0, 1) isteği gelirse istek matrisi yandaki gibi olur ve **sistem deadlock olabilir. P₀ tümünü bıraksa bile!!!**
- **P₀ tümünü bıraksa bile**, kalan kaynak miktarı, **P₁, P₂, P₃ ve P₄ için yeterli değildir** ve deadlock oluşur.

| | <u>Request</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 |

39

Konular

- Sistem modeli
- Deadlock tanımı ve özellikleri
- Deadlock yönetimi için metotlar
- Deadlock önleme
- Deadlock'tan kaçınma
- Deadlock algılama
- **Deadlock'tan kurtulma**

40

Deadlock'tan kurtulma

Process termination

- Sistemde deadlock oluştuğunda **operatör tarafından manuel giderilebilir.**
- **Sistem otomatik olarak deadlock recover yapabilir.**
- Process termination yönteminde 2 tür çözüm olabilir:
 - **Tüm deadlock durumundaki process'ler sonlandırılır.** Process'lerin sonuçları kaybolabilir.
 - **Deadlock döngüsü ortadan kalkıncaya kadar her adımda bir deadlock process sonlandırılır.** Her process sonlandığında deadlock cycle kontrolü yapılması gereklidir.
- Bir process'in **abort edilmesi sonucunda tutarsızlıklar ortaya çıkabilir.**
 - Dosya yazma işlemi yarıda kalabilir.
 - Yazıcıda devam eden yazdırma işlemi yarıda kalabilir.

45

Deadlock'tan kurtulma

Resource preemption

- **Deadlock cycle ortadan kalkıncaya kadar process'lere atanmış kaynaklar alınarak başka process'lere atanır.**
 - **Selecting a victim:** **Kaynak veya process seçiminde maliyet hesaplanabilir.** Deadlock olma sıklığı, çalışma süresi, vb.
 - **Rollback:** **Bir process'ten bir kaynağı alınca, normal çalışmasına devam edip etmeyeceğine karar verilir.** Buna karar vermek zor olduğundan process abort edilir ve yeniden başlatılır.
 - **Starvation:** **Eğer bir process sürekli kesiliyorsa görevini hiçbir zaman tamamlayamayabilir (starvation).** Bir process'in victim olarak seçilmesinde maksimum limit belirlenebilir.

46