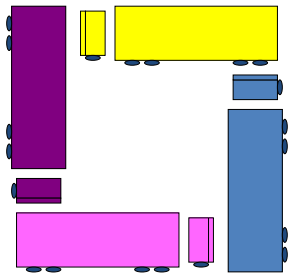
Bilgisayar İşletim Sistemleri
BLG 312

Ölümcül Kilitlenme

Ölümcül Kilitlenme

- Sistem kaynaklarını ortak olarak kullanan veya birbirleri ile haberleşen bir grup prosesin kalıcı olarak bloke olması durumu
- Örnek: proseslerin ellerindeki kaynakları bırakmadan başka kaynak istemeleri ölümcül kilitlenmeye neden olabilir
- Tanım: Bir proses kümesindeki **her** prosesin, kümede yer alan bir başka prosesin yaratabileceği bir durumu bekliyor olması, kümede yer alan prosesler için ölümcül kilitlenme ile sonuçlanır

Gerçek hayatta kilitlenme-trafik☺



Proseslerin Kaynak Kullanımı

- Prosesler kaynak edinmek istediklerinde izlenen yol şöyledir:

1. Kaynak iste – derhal elde edilemez ise, proses kaynak hazır olana kadar bloke olur
2. Kaynağı kullan
3. Kaynağı iade et

Proseslerin Kaynak Kullanımı

- kaynak isteklerinin karşılanamaması durumunda uygulanabilecek farklı yaklaşımlar:
  - İşletim sistemi prosesi bloke eder, kaynak hazır olunca prosesi uyandırır
  - İstek çağrısı prosese bir hata kodu ile döner, proses gerekirse bir süre sonra tekrar dener

Ölümcül Kilitlenme

P1	**P2**
iste(D);	iste(T);
kilitle(D);	kilitle(T);
iste(T);	iste(D);
kilitle(T);	kilitle(D);
<işlem yap>	<işlem yap>
kilit_aç(T);	kilit_aç(D);
kilit_aç(D);	kilit_aç(T);

: çalışmanın kesintiye uğrama noktası →

Ölümcül kilitlenme potansiyeli var!

Ölümcül Kilitlenme

Örnek: Sistemde toplam 200K sekizli boyunda bellek bölgesi proseslere atanabilir durumda. Bellek kullanan başka proses olmadığı varsayımıyla, aşağıdaki istekler oluşursa ölümcül kilitlenme potansiyeli var.

P1	**P2**
iste(80K);	iste(70K);
...	...
iste(60K);	iste(80K);
...	...

Elde 50K kaldı, diğer isteklerin karşılanma olasılığı yok

Ölümcül Kilitlenme

Örnek: Mesaj alma komutu bloke olan türden ise (proses mesaj gelene kadar bloke olur) aşağıda verilen kod ölümcül kilitlenme ile sonuçlanır:

P1	**P2**
mesaj_al(P2);	mesaj_al(P1);
...	...
mesaj_gönder(P2);	mesaj_gönder(P1);
...	...

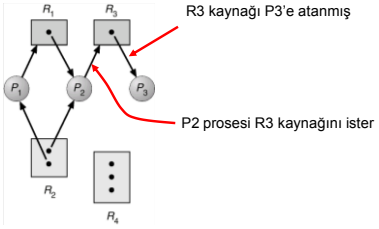
Ölümcül Kilitlenmenin Belirlenmesi

- Ölümcül kilitlenmenin var olduğu veya olası olduğu bir graf kullanarak gösterilebilir. Grafta düğümler ve ayrıtlar yer alır:
  - düğümler:
    - ○ daire: prosesler (P1, P2,..,Pn)
    - □ kare: kaynaklar (K1, K2,..,Kn)
  - ayrıtlar: (Pi, Kj) veya (Ki, Pj)
    - proses → kaynak : prosesin kaynak isteği

    - kaynak → proses : kaynak prosese atanmış

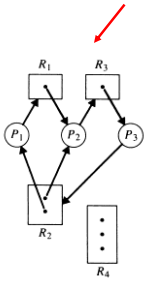
Kaynak Atama Grafı

- Eğer grafta bir çevrim yok ise, ölümcül kilitlenme yoktur.
- Eğer bir çevrim var ise:
  a) Eğer kaynaktan sadece 1 adet var ise, kilitlenme oluşmuştur.
  b) Eğer kaynaktan bir kaç tane var ise, kilitlenme olasılığı vardır

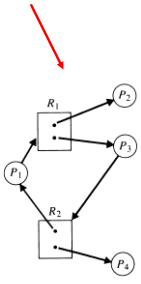


Kaynak Atama Grafı

Ölümcül kilitlenme mevcut



Graf çevrim içerir ancak kilitlenme yok



Ölümcül Kilitlenmeye Neden Olan Durumlar

- karşılıklı dışlama koşulu
  - paylaşılan kaynaklar üzerinde bir anda sadece bir prosesin iş yapması
- sahiplenme ve bekleme koşulu
  - elindeki kaynağı bırakmadan başka kaynak isteyen ve alamazsa beklemeye başlayan prosesler olması
- geri alınamaz kaynak koşulu
  - bir prosese atanan kaynakların işletim sistemi tarafından prosesin isteği dışında elinden alınamıyor olması
- çevrel bekleme koşulu
  - iki veya daha fazla prosesten oluşan, her bir prosesin bir öncekinin elinde tuttuğu kaynağı istediği bir çevrel kuyruğun oluşması

Ölümcül Kilitlenme

- ölümcül kilitlenmeye karşı kullanılan yaklaşımlar
  - sistemi ölümcül kilitlenme hiç olmayacak şekilde tasarlama
    - ölümcül kilitlenmeye neden olan koşulları (bir önceki sayfa) ortadan kaldırma
  - ölümcül kilitlenmeyi sezme
    - Kilitlenmenin oluştuğunu fark edip, çözülene kadar tarafları ortadan kaldırma
  - ölümcül kilitlenmeyi önleme (kaçınma)
    - istekleri kilitlenmeye neden olabilecek prosesleri başlatmama
    - kilitlenmeye neden olabilecek istekleri karşılamama

Ölümcül Kilitlenmeyi Önleme

- Ölümcül kilitlenmeyi önlemeye yönelik algoritma: Banker algoritması
  - Dijkstra, 1965
  - sistemde sabit sayıda kaynak ve proses vardır
  - sistemin durumu: bir anda hangi kaynakların hangi proseslere atanmış olduğu bilgisi ve proseslerin geriye kalan kaynak istekleri
  - Sistem durumu bir dizi veri yapısı ile temsil edilir:
    - *kaynak* ve *boş* vektörleri
    - *atanmış* ve *max_istek* matrisleri

Banker Algoritması-değişkenler

- *kaynak*: sistemde var olan toplam kaynak sayısı
- *boş*: kullanılabilir durumda olan kaynaklar ve sayıları
- *atanmış*: hangi kaynaktan, kaç adet, hangi prosese atanmış olduğu bilgisi
- *max_istek*: proseslerin yaşamları boyunca isteyebilecekleri en yüksek kaynak sayısı

Banker Algoritması

- Güvenli durum: ölümcül kilitlenmeye yol açmadan tüm proseslerin çalışıp sonlanmalarına imkan veren bir kaynak atama sekansının var olma durumu
  - bu durum tüm proseslerin sonlanmasını garantiler
- Güvensiz durum: tüm proseslerin sonlanmasına izin verecek bir kaynak atama sekansının var olmaması
  - bazı prosesler için sonlanamama olasılığı söz konusudur

Güvenli Durum Örneği

- Toplam kaynak sayısı: 12

	max istek	halen sahip olunan	kalan istek sayısı
p0	10	5	5
P1	4	2	2
p2	9	2	7

kalan kaynak sayısı:3

- Bu durum güvenlidir çünkü tüm isteklerin karşılanabileceği bir proses sekansı mevcuttur: <p1, p0, p2>
  - p1 var olan kaynakları kullanarak sonlanabilir → kaynak sayısı: 5
  - p0 var olan kaynaklar + p1 kaynakları kullanarak sonlanabilir
  - p2 var olan kaynaklar + p1+p0 kaynakları kullanarak sonlanabilir
- Eğer p2 bir kaynak isterse, güvensiz duruma geçişi önlemek için beklemeli

Güvensiz Durum Örneği

Örnek 2: Toplam 12 kaynak var.

Güvensiz durum X

- ölümcül kilitlenme olmayacağı garanti edilemez!

Proses	Sahip	Max_ İstek	Kalan_ İstek
A	8	10	2
B	2	5	3
C	1	3	2

Atanmış=11 *Boş*=1

Güvenli Durum Örneği

Örnek 1: Toplam 12 kaynak var.

Güvenli durum √

- önce B'ye 2 kaynak ver
- B bitince 6 kaynak boş
- A ve C de sonlanır.

Proses	Sahip	Max_ İstek	Kalan_ İstek
A	1	4	3
B	4	6	2
C	5	8	3

Atanmış=10 *Boş*=2

Güvenli → Güvensiz Geçiş Örneği

Toplam 12 kaynak var.

C prosesi bir kaynak daha isterse bu istek karşılanır mı?

- istenen kaynak verilmiş gibi sistemin durumunu gösteren vektör ve matrisler güncellenir → güvensiz durum oluşur
- C prosesinin kaynak isteği karşılanmaz!

Güvenli durum

Proses	Sahip	Max_ İstek	Kalan_ İstek
A	1	4	3
B	4	6	2
C	5	8	3

Atanmış=10 *Boş*=2

Güvensiz durum

Proses	Sahip	Max_ İstek	Kalan_ İstek
A	1	4	3
B	4	6	2
C	6	8	2

Atanmış=11 *Boş*=1

Banker Algoritması – Tek Tip Kaynak

Verilerin oluşturulması:

```
kalan_birim=toplam_birim;
for i= 1 to proses_sayısı do
  begin
  kalan_birim=kalan_birim –sahip[i];
  bitemeyebilir[i]=TRUE;
  kalan_istek[i]=max_istek[i]sahip[i];
end;
```

Banker Algoritması – Tek Tip Kaynak

```
bayrak=TRUE;
while (bayrak) do
begin
  bayrak=FALSE;
  for i=1 to proses_sayısı do
  begin
    if ((bitemeyebilir[i]) AND
     (kalan_istek[i]<= kalan_birim)) then
    begin
      bitemeyebilir[i]=FALSE;
      kalan_birim=kalan_birim+sahip[i];
      bayrak=TRUE;
    end;
  end;
end;
```

Banker Algoritması – Tek Tip Kaynak

Karar aşaması:

```
if (kalan_birim == toplam_birim)
then
   -----GÜVENLİ DURUM------
else
   -----GÜVENSİZ DURUM-----
```

Banker Algoritması – Çoklu Kaynak Durumu

Max. İstek Matrisi

	K1	K2	K3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

Atanmış Matrisi

	K1	K2	K3
P1	1	0	0
P2	6	1	2
P3	2	1	1
P4	0	0	2

Kalan İstek Matrisi

	K1	K2	K3
P1	2	2	2
P2	0	0	1
P3	1	0	3
P4	4	2	0

K1	K2	K3
9	3	6

Kaynak Vektörü

K1	K2	K3
0	1	1

Boş Kaynak Vektörü

P2, P3, P1 veya P4 sırası ile prosesler çalışırsa tümü sonlanaibilir.

⇒ GÜVENLİ DURUM √

Banker Algoritması – Çoklu Kaynak Durumu

Soru: Örnekteki güvenli durumdayken P3 prosesi bir adet daha K3 kaynağından isterse bu istek karşılanır mı?

⇒ **Sonuç:** Bu kaynak isteğinin karşılanması güvensiz durum oluşturur. İstek karşılanmaz!

Banker Algoritması – Çoklu Kaynak Durumu

Max. İstek Matrisi

	K1	K2	K3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

Atanmış Matrisi

	K1	K2	K3
P1	1	0	0
P2	6	1	2
P3	2	1	2
P4	0	0	2

Kalan İstek Matrisi

	K1	K2	K3
P1	2	2	2
P2	0	0	1
P3	1	0	2
P4	4	2	0

K1	K2	K3
9	3	6

Kaynak Vektörü

K1	K2	K3
0	1	0

Boş Kaynak Vektörü

Sadece bir K2 ile hiç bir proses sonlanamaz

⇒ GÜVENSİZ DURUM √

Banker Algoritması – Algoritmanın Uygulanması

1. kalan_istek matrisinde ≤ boş_vektörü olan satır var mı?
   yoksa ölümcül kilitlenme olabilir: *Güvensiz* durum
2. 1. adımda bulunan prosesin çalışıp tüm kaynaklarını iade ettiğini varsay, kaynakları boş_vektörüne ekle, prosesi sonlandı olarak işaretle
3. 1. ve 2. adımları tüm prosesler sonlanana kadar tekrarla (sonlanmayan proses varsa *Güvensiz* durum)

Banker Algoritması

- algoritmayı uygulayabilmek için:
  - prosesler çalışma başında tüm kaynak isteklerini bildirmeli
  - proses ve kaynak sayısı sabit olmalı
  - proseslerin koşma sıralarının önemi olmamalı
  - elinde kaynak tutan proses kaynaklarını bırakmadan sonlanmamalı
- algoritma en kötü durumu göz önüne alıp istekleri karşılar / karşılamaz
  - ölümcül kilitlenmeye neden olabileceği için geri çevrilen istek aslında ölümcül kilitlenme oluşturmayabilir ⇒ etkin olmayan kaynak kullanımı!
- algoritma her kaynak isteği olduğunda çalışır ⇒ maliyeti yüksek

Ölümcül Kilitlenmeyi Sezme

- önleme yöntemleri kadar kısıtlayıcı değil
- tüm kaynak istekleri karşılanır
- belirli aralıklarla sistemde ölümcül kilitlenme olup olmadığı kontrol edlir
  - ölümcül kilitlenme olduysa:
    - kilitlenen tüm prosesleri sonlandır
    - kilitlenme ortadan kalkana kadar prosesleri tek tek sonlandır
    - ...
- her kaynak isteğinde çalıştırılmadığından maliyeti daha düşük
- daha etkin kaynak kullanımı
- hangi sıklıkta kontrol yapılacağı sistemde ölümcül kilitlenme olma sıklığına göre ayarlanır