1906003052015

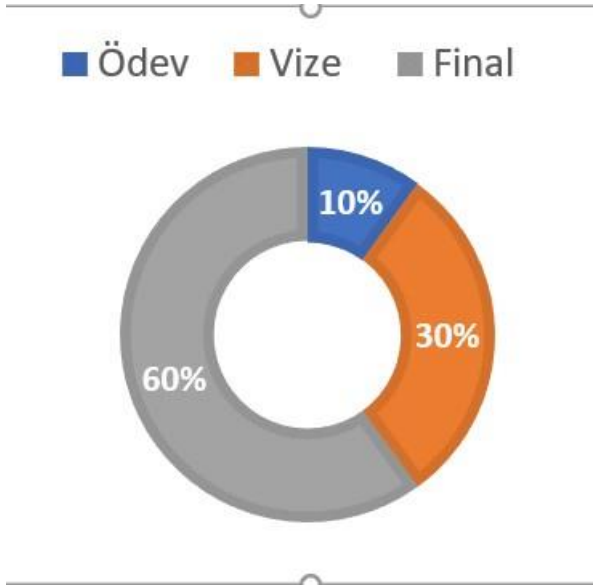
İşletim Sistemleri

Dr. Öğr. Üy. Önder EYECİOĞLU
Bilgisayar Mühendisliği

Giriş

Ders Günü ve Saati:
Çarşamba: 13:00-16:00

- Uygulama Unix (Linux) İşletim sistemi
- Devam zorunluluğu %70
- Uygulamalar C programlama dili üzerinde gerçekleştirilecektir. Öğrencilerden programlama bilgisi beklenmektedir.



HAFTA	KONULAR
Hafta 1	: İşletim sistemlerine giriş, İşletim sistemi stratejileri
Hafta 2	: Sistem çağrıları
Hafta 3	: Görev, görev yönetimi
Hafta 4	: İplikler
Hafta 5	: İş sıralama algoritmaları
Hafta 6	: Görevler arası iletişim ve senkronizasyon
Hafta 7	: Semaforlar, Monitörler ve uygulamaları
Hafta 8	: Vize
Hafta 9	: Kritik Bölge Problemleri
Hafta 10	: Kilitlenme Problemleri
Hafta 11	: Bellek Yönetimi
Hafta 12	: Sayfalama, Segmentasyon
Hafta 13	: Sanal Bellek
Hafta 14	: Dosya sistemi, erişim ve koruma mekanizmaları, Disk planlaması ve Yönetimi
Hafta 15	: Final

KLASİK SENKRONİZASYON PROBLEMLERİ

Semafor, Mutual Exclusion dışında diğer senkronizasyon problemlerinde de kullanılabilir.

Aşağıda, işbirliği yapan süreçlerin mevcut olduğu sistemlerde süreç senkronizasyonunun kusurlarını gösteren klasik problemlerden bazıları verilmiştir.

Aşağıdaki üç sorunu tartışacağız:

1.Sınırlı Tampon (Üretici-Tüketici) Problemi

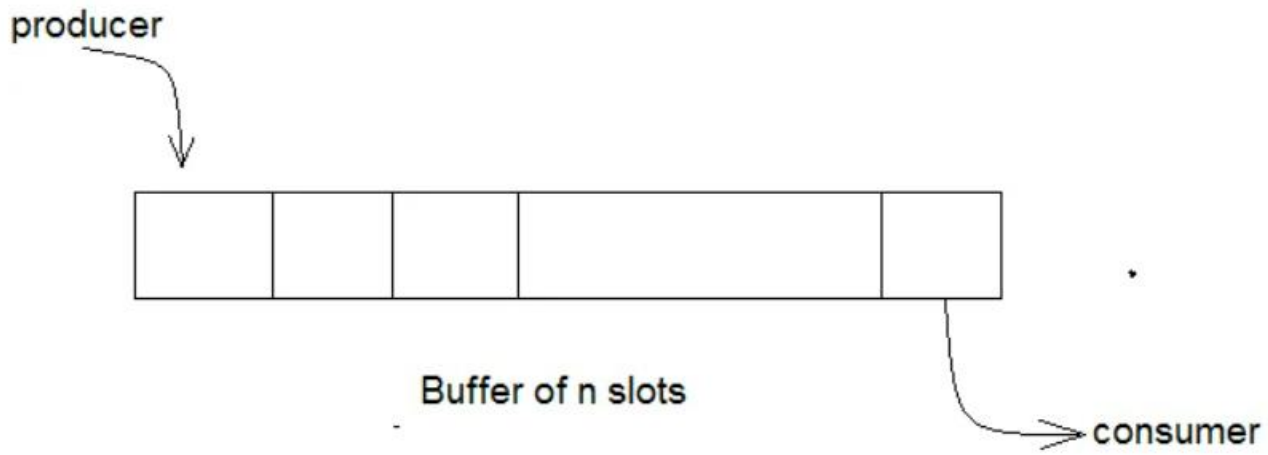
2.Filozofları Doyurma Problemi

3.Okur Yazarlar Problemi

SINIRLI TAMPON PROBLEMİ

Üretici tüketici problemi olarak da adlandırılan sınırlı tampon problemi, senkronizasyonun klasik problemlerinden biridir.

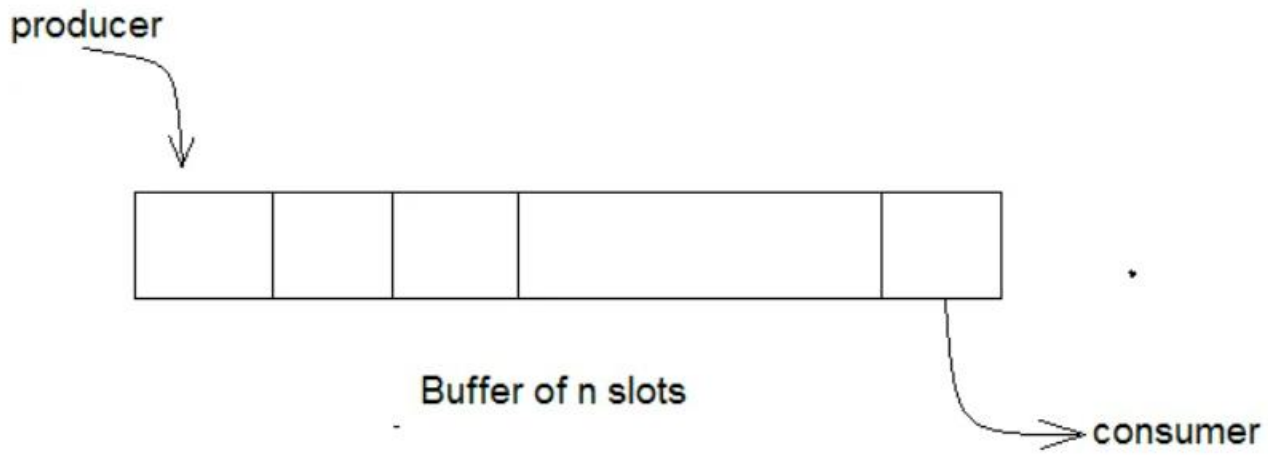
Bir n yuva arabelleği vardır ve her yuva bir birim veri depolama kapasitesine sahiptir. Arabellek üzerinde çalışan, yani **üretici** ve **tüketici** olmak üzere iki süreç vardır .



SINIRLI TAMPON PROBLEMİ

Bir üretici, arabelleğin boş bir yuvasına veri eklemeye çalışır. Bir tüketici, arabellekteki doldurulmuş bir yuvadan verileri kaldırmaya çalışır. Şimdiye kadar tahmin edebileceğiniz gibi, bu iki süreç aynı anda yürütülüyorsa beklenen çıktıyı üretmeyecektir.

Üretici ve tüketicinin bağımsız bir şekilde çalışmasını sağlamanın bir yolu olmalı.



SINIRLI TAMPON PROBLEMİ

Bu sorunun bir çözümü semafor kullanmaktır. Burada kullanılacak semaforlar şunlardır:

- m, kilidi almak ve serbest bırakmak için kullanılan **ikili bir semafor** .
- empty, başlangıçta tüm yuvalar boş olduğundan, başlangıç değeri arabellekteki yuva sayısı olan bir **sayma semaforudur** .
- full, başlangıç değeri 0 olan bir **sayma semaforu** .

Herhangi bir anda, boş değeri, arabellekteki boş yuvaların sayısını temsil eder ve dolu, arabellekteki dolu yuvaların sayısını temsil eder.

SINIRLI TAMPON PROBLEMİ

ÜRETİCİ

kopyala

```
do
{
    // wait until empty > 0 and then decrement 'empty'
    wait(empty);
    // acquire lock
    wait(mutex);

    /* perform the insert operation in a slot */

    // release lock
    signal(mutex);
    // increment 'full'
    signal(full);
}
while(TRUE)
```

SINIRLI TAMPON PROBLEMİ

ÜRETİCİ

- Bir üretici için yukarıdaki koda baktığımızda, bir üreticinin önce en az bir boş slot olana kadar beklediğini görebiliriz.
- Ardından boş semaforu azaltır, çünkü üretici bu yuvalardan birine veri ekleyeceğinden artık bir boş yuva daha az olacaktır.
- Ardından, arabellek üzerinde kilit alır, böylece üretici işlemini tamamlayana kadar tüketici arabelleğe erişemez.
- Ekleme işlemini gerçekleştirdikten sonra , üretici arabellekteki bir yuvayı doldurduğu için kilit serbest bırakılır ve dolu değeri artırılır.

SINIRLI TAMPON PROBLEMİ

TÜKETİCİ

```
do
{
    // wait until full > 0 and then decrement 'full'
    wait(full);
    // acquire the lock
    wait(mutex);

    /* perform the remove operation in a slot */

    // release the lock
    signal(mutex);
    // increment 'empty'
    signal(empty);
}
while(TRUE);
```

SINIRLI TAMPON PROBLEMİ

TÜKETİCİ

- Tüketici, arabellekte en az bir tam yuva olana kadar bekler.
- Daha sonra tam semaforu azaltır çünkü tüketici işlemi tamamladıktan sonra dolu yuva sayısı birer birer azalacaktır.
- Bundan sonra, tüketici arabellek üzerinde kilit elde eder.
- Bunu takiben tüketici, dolu yuvalardan birindeki verilerin kaldırılması için kaldırma işlemini tamamlar.
- Ardından, tüketici kilidi serbest bırakır.
- Son olarak, boş semafor 1 artırılır, çünkü tüketici dolu bir yuvadan veriyi yeni çıkarmış ve böylece boş bırakmıştır.

FİLOZOFLARI DOYURMA PROBLEMİ

1965'te Dijkstra, yemek filozofları problemi adını verdiği bir senkronizasyon problemini ortaya koydu ve sonra çözdü.

Filozofları doyurma sorunu, sınırlı kaynakların bir grup sürece kilitlenmeden ve açlıktan arınmış bir şekilde tahsis edilmesini içerir.

FİLOZOFLARI DOYURMA PROBLEMİ

Sorun oldukça basit bir şekilde şu şekilde ifade edilebilir. Beş filozof yuvarlak bir masanın etrafında oturuyor. Her filozofun bir tabak spagetti vardır. Spagetti o kadar kaygandır ki bir filozofun onu yemek için iki çatala ihtiyacı vardır. Her plaka çifti arasında bir çatal bulunur.

FİLOZOFLARI DOYURMA PROBLEMİ

Bir filozofun yaşamı, birbirini izleyen yeme ve düşünme dönemlerinden oluşur. (Bu, filozoflar için bile bir tür soyutlamadır, ancak diğer faaliyetler burada önemsizdir.) Bir filozof yeterince acıktığında, sol ve sağ çatallarını birer birer, her iki sırada da elde etmeye çalışır. İki çatal almayı başarırsa, bir süre yemek yer, ardından çatalları bırakır ve düşünmeye devam eder. Anahtar soru şudur: Her filozof için yapması gerekeni yapan ve asla takılıp kalmayan bir program yazabilir misiniz?

FİLOZOFLARI DOYURMA PROBLEMİ

Take fork, belirtilen fork kullanılabilir duruma gelene kadar bekler ve ardından onu alır. Ne yazık ki, bariz çözüm yanlış. Beş filozofun hepsinin aynı anda sol çatallarını aldıklarını varsayalım. Hiçbiri doğru çatalını alamayacak ve bir çıkmaza girecek.

FİLOZOFLARI DOYURMA PROBLEMİ

```
#define N 5                                     /* number of philosophers */

void philosopher(int i)                         /* i: philosopher number, from 0 to 4 */
{
        while (TRUE) {
                think( );                       /* philosopher is thinking */
                take_fork(i);                   /* take left fork */
                take_fork((i+1) % N);           /* take right fork; % is modulo operator */
                eat( );                         /* yum-yum, spaghetti */
                put_fork(i);                    /* put left fork back on the table */
                put_fork((i+1) % N);            /* put right fork back on the table */
        }
}
```

FİLOZOFLARI DOYURMA PROBLEMİ

Programı kolayca değiştirebiliriz, böylece sol çatalı aldıktan sonra program doğru çatalın mevcut olup olmadığını kontrol eder. Değilse, filozof solu bırakır, bir süre bekler ve ardından tüm süreci tekrarlar. Bu öneri de, farklı bir nedenle de olsa başarısız olur. Biraz şanssızlıkla, tüm filozoflar algoritmayı aynı anda başlatabilir, sol çatallarını alıp, sağ çatallarının müsait olmadığını görerek, sol çatallarını bırakabilir, bekleyebilir, sol çatallarını aynı anda tekrar alabilir ve böylece bu süreç sonsuza kadar devam edebilir.

FİLOZOFLARI DOYURMA PROBLEMİ

Tüm programların süresiz olarak devam ettiği ancak herhangi bir ilerleme sağlayamadığı böyle bir duruma açlık (starvation) denir.

Problemde karşılaşılan diğer bir sorun ise ölümcül kilitlenmedir (deadlock). Yanlış bir tasarım sonucunda, tek çatal alan ve çatalı bırakmak için diğer filozofun bırakmasını bekleyen bir filozof sistemi kilitleyebilir. Bu da problemde bulunan ikinci risktir.

FİLOZOFLARI DOYURMA PROBLEMİ

Semafor Çözümü

Basit bir çözüm, her bir çubuğu bir semaforla temsil etmektir. Bir filozof, o semafor üzerinde bir wait() işlemi yürüterek bir çubuk yakalamaya çalışır. Uygun semaforlarda sinyal() işlemini yürüterek yemek çubuklarını serbest bırakır.

```
semaphore chopstick [5];
```

FİLOZOFLARI DOYURMA PROBLEMİ

Semafor Çözümü

Rastgele bir filozof i'nin yapısı

```
while (true) {
   wait(chopstick[i]);
   wait(chopstick[(i+1) % 5]);
      . . .
   /* eat for a while */
      . . .
   signal(chopstick[i]);
   signal(chopstick[(i+1) % 5]);
      . . .
   /* think for awhile */
      . . .
}
```

FİLOZOFLARI DOYURMA PROBLEMİ

Semafor Çözümü

Bu çözüm, iki komşunun aynı anda yemek yememesini garanti etse de, bir kilitlenme yaratabileceğinden yine de reddedilmelidir. Beş filozofun hepsinin aynı anda acıktığını ve her birinin sol yemek çubuğunu tuttuğunu varsayalım. Çubuğun tüm öğeleri şimdi 0'a eşit olacaktır. Her filozof sağ çubuk çubuğunu almaya çalıştığında, sonsuza kadar ertelenecektir.

FİLOZOFLARI DOYURMA PROBLEMİ

Semafor Çözümü

Kilitlenme sorununa birkaç olası çözüm aşağıdaki gibidir:

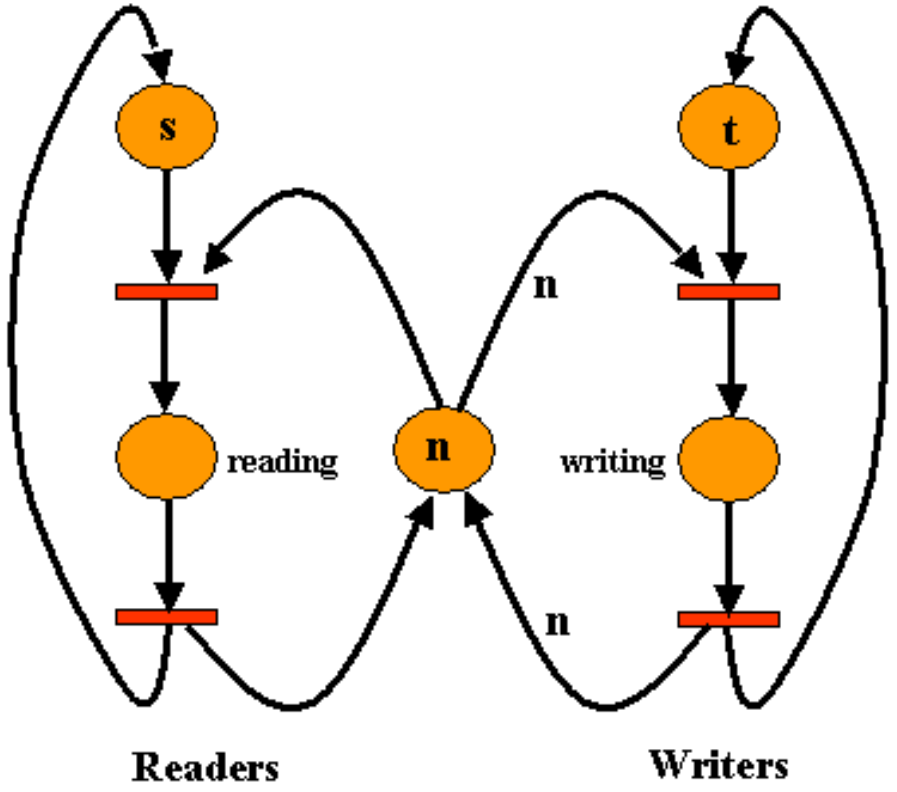
• Masada aynı anda en fazla dört filozofun oturmasına izin verin.

• Bir filozofun yemek çubuklarını yalnızca her iki çubuk da mevcutsa almasına izin verin (bunu yapmak için onları kritik bir bölümden alması gerekir).

• Asimetrik bir çözüm kullanın—yani, tek sayılı bir filozof önce sol çubuk çubuğunu ve sonra sağ çubuk çubuğunu alırken, çift sayılı bir filozof sağ çubuk çubuğunu ve sonra sol çubuk çubuğunu alır

OKUYUCU YAZICI PROBLEMİ

Yemek felsefecileri sorunu, G/Ç aygıtları gibi sınırlı sayıda kaynağa özel erişim için rekabet eden süreçleri modellemek için kullanışlıdır. Bir diğer ünlü problem, bir veri tabanına erişimi modelleyen okuyucular ve yazarlar problemidir (Courtois ve diğerleri, 1971). Örneğin, okumak ve yazmak isteyen birbiriyle rekabet halindeki birçok süreci olan bir havayolu rezervasyon sistemi düşünün. Veritabanını aynı anda okuyan birden fazla işlemin olması kabul edilebilir, ancak bir işlem veritabanını güncelliyorsa (yazıyorsa), başka hiçbir işlemin veritabanına erişimi olmayabilir, hatta okuyucular bile. Soru şu ki, okuyucuları ve yazarları nasıl programlıyorsunuz?

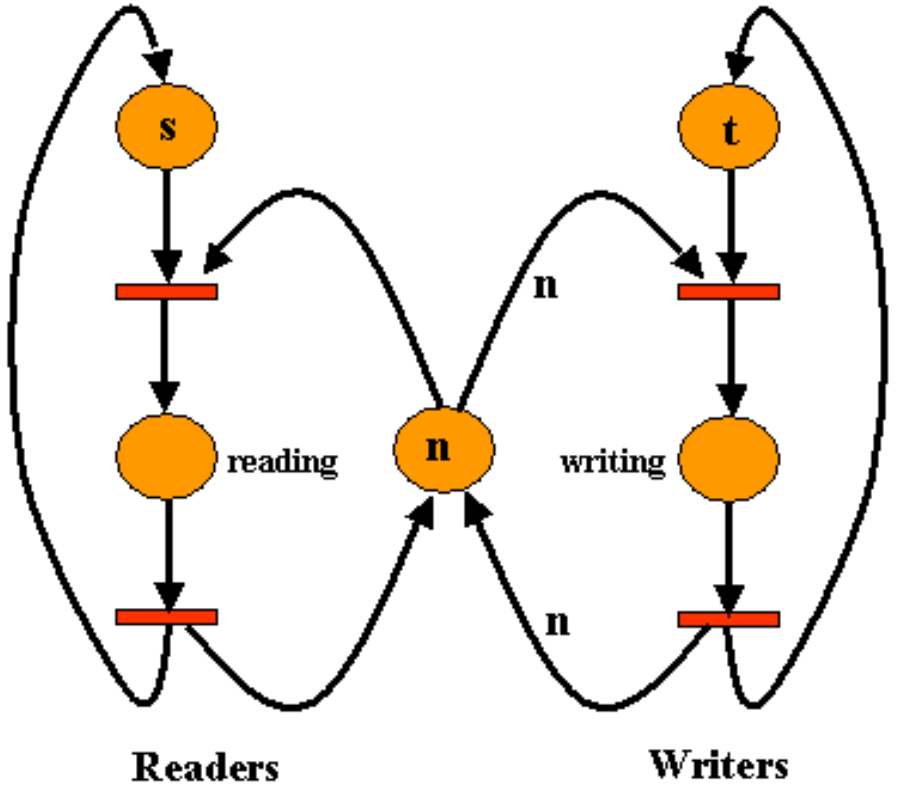


OKUYUCU YAZICI PROBLEMİ

Bu problemde , sadece paylaşılan verileri okuyan ve asla değiştirmeyen bazı işlemler (okuyucu olarak adlandırılır) vardır ve verileri okumaya ek olarak veya okumak yerine değiştirebilen başka işlemler (yazıcılar olarak adlandırılır) vardır.

Çoğu okuyucu ve yazarların göreli önceliklerine odaklanan çeşitli türde okuyucu-yazar sorunu vardır.

Bu problemdeki ana karmaşıklık, birden fazla okuyucunun aynı anda verilere erişmesine izin vermekten kaynaklanmaktadır.

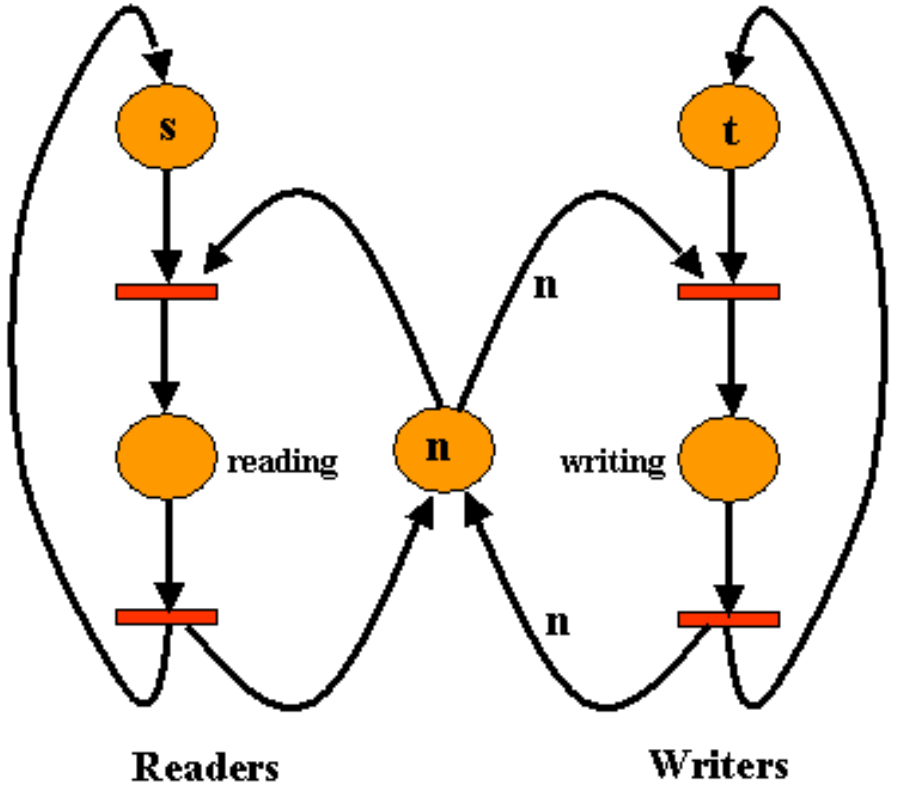


OKUYUCU YAZICI PROBLEMİ

Okuyucular/yazarlar sorunu şu şekilde tanımlanır:

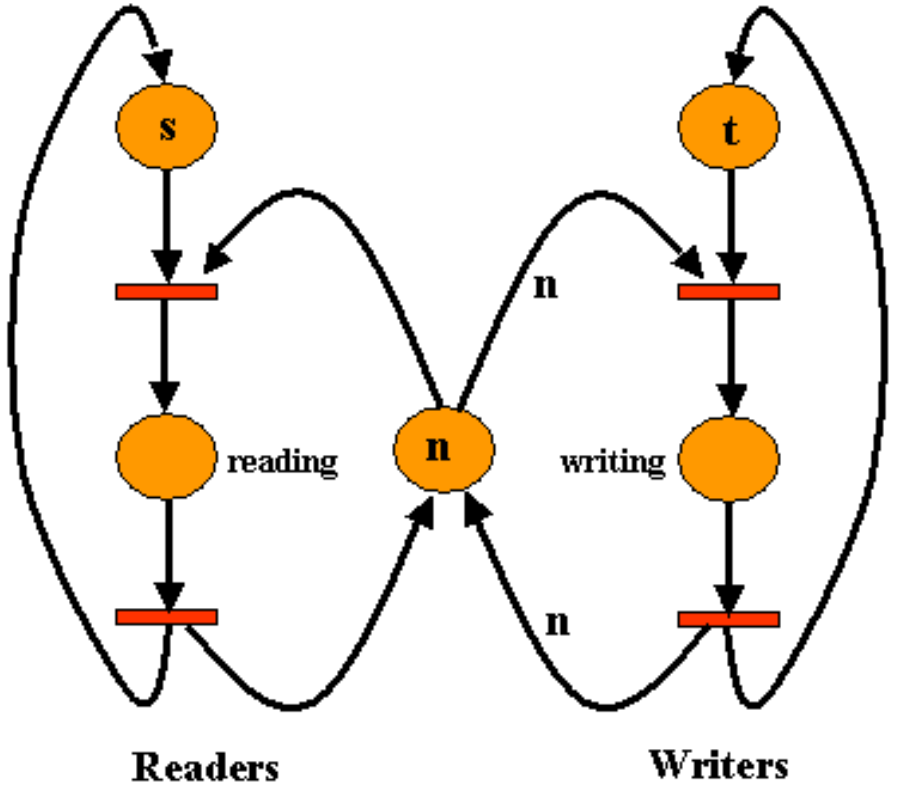
Bir dizi işlem arasında paylaşılan bir veri alanı vardır. Veri alanı bir dosya, bir ana bellek bloğu veya hatta bir işlemci kayıtları bankası olabilir. Yalnızca veri alanını okuyan (okuyucular) ve yalnızca veri alanına yazan (yazıcılar) bir dizi işlem vardır. Sağlanması gereken koşullar aşağıdaki gibidir:

1. Herhangi bir sayıda okuyucu dosyayı aynı anda okuyabilir.
2. Dosyaya aynı anda yalnızca bir yazar yazabilir.
3. Bir yazar dosyaya yazıyorsa, hiçbir okuyucu onu okuyamaz.



OKUYUCU YAZICI PROBLEMİ

Okur-yazar sorununun, tümü öncelikleri içeren çeşitli varyasyonları vardır. İlk okuyucu-yazar problemi olarak adlandırılan en basit problem, bir yazar paylaşılan nesneyi kullanmak için zaten izin almadıkça hiçbir okuyucunun bekletilmemesini gerektirir. Başka bir deyişle, hiçbir okuyucu, bir yazar bekliyor diye diğer okuyucuların bitirmesini beklememelidir. İkinci okuyucu-yazar sorunu, bir yazar hazır olduğunda, o yazarın yazısını bir an önce gerçekleştirmesini gerektirir. Başka bir deyişle, bir yazar nesneye erişmeyi bekliyorsa, hiçbir yeni okuyucu okumaya başlayamaz.

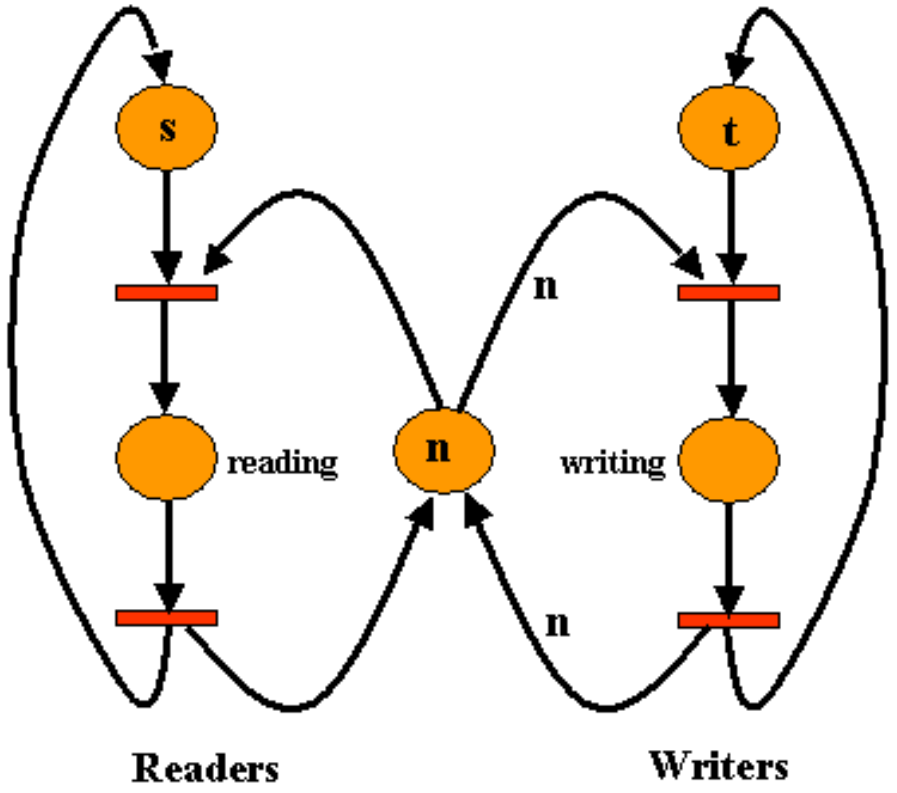


OKUYUCU YAZICI PROBLEMİ

Çözüm

İlk okuyucu-yazar sorununun çözümünde, okuyucu süreçleri aşağıdaki veri ya

```
semaphore rw_mutex = 1;
semaphore mutex = 1;
int read_count = 0;
```



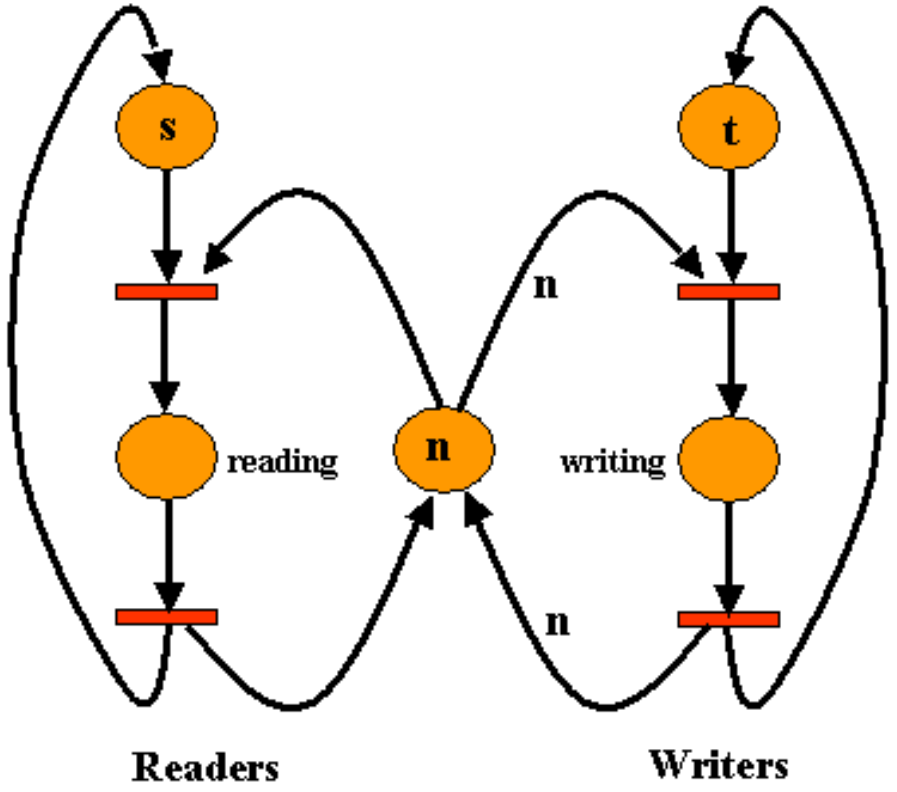
$mutex$ ve rw_mutex ikili semaforları 1 olarak başlatılır; $read\_count$, 0 olarak başlatılan bir sayma semaforudur. rw_mutex semaforu hem okuyucu hem de yazar süreçlerinde ortaktır.

OKUYUCU YAZICI PROBLEMİ

Değişken okuma sayısı güncellendiğinde karşılıklı dışlamayı sağlamak için $mutex$ semaforu kullanılır.

$read\_count$ değişkeni, nesneyi şu anda kaç işlemin okuduğunu takip eder.

rw_mutex semeforu, yazarlar için ortak dışlama semaforu olarak işlev görür. Kritik bölüme giren veya çıkan ilk veya son okuyucu tarafından da kullanılır. Diğer okuyucular kritik bölümlerindeyken giren veya çıkan okuyucular tarafından kullanılmaz.



OKUYUCU YAZICI PROBLEMİ

```
while (true) {
   wait(rw_mutex);
      . . .
   /* writing is performed */
      . . .
   signal(rw_mutex);
}
```

Figure 7.3 The structure of a writer process.

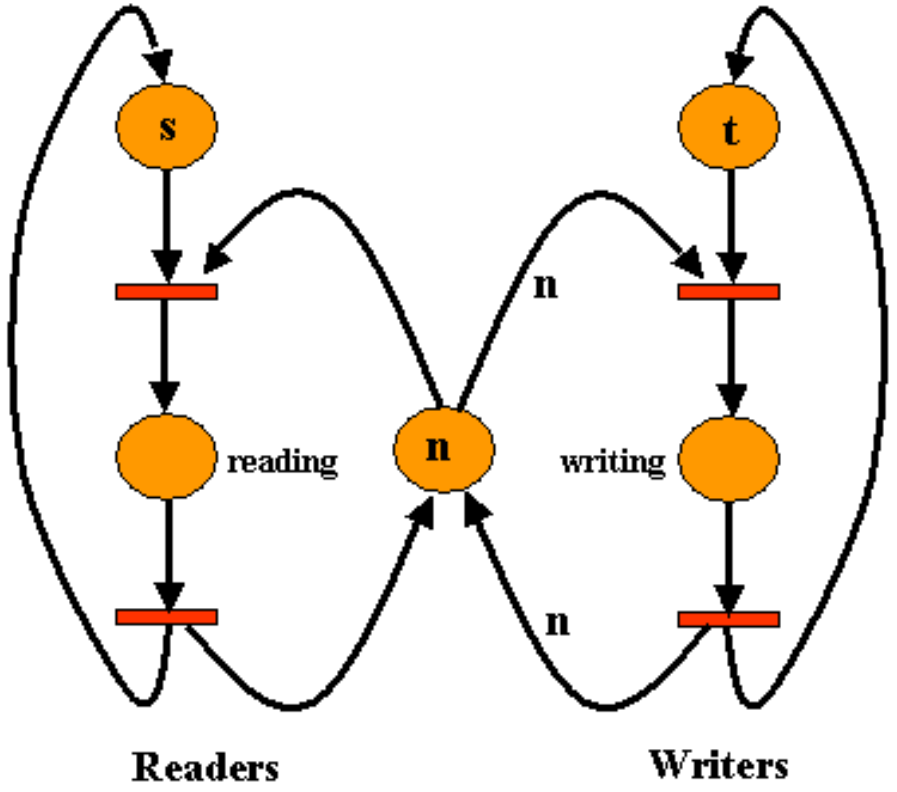
OKUYUCU YAZICI PROBLEMİ

```
while (true) {
   wait(mutex);
   read_count++;
   if (read_count == 1)
      wait(rw_mutex);
   signal(mutex);
      . . .
   /* reading is performed */
      . . .
   wait(mutex);
   read_count--;
   if (read_count == 0)
      signal(rw_mutex);
   signal(mutex);
}
```

Figure 7.4 The structure of a reader process.

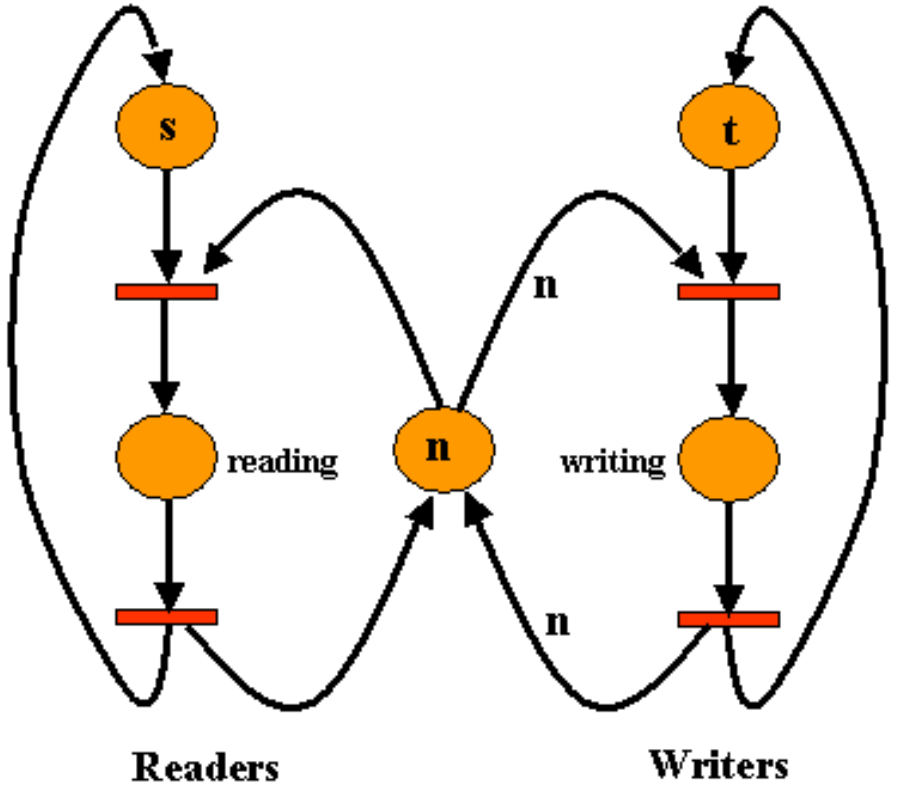
OKUYUCU YAZICI PROBLEMİ

Bir yazar kritik bölümdeyse ve n okuyucu bekliyorsa, o zaman rw muteksinde bir okuyucunun kuyruğa alındığını ve muteks üzerinde $n-1$ okuyucunun kuyruğa alındığını unutmayın. Ayrıca, bir yazar, $signal(rw\ mutex)$ yürüttüğünde, bekleyen okuyucuların veya tek bir bekleyen yazarın yürütülmesine devam edebileceğimizi gözlemleyin. Seçim, zamanlayıcı tarafından yapılır.



OKUYUCU YAZICI PROBLEMİ

Okur-yazar sorunu ve çözümleri, bazı sistemlerde okuyucu-yazar kilitleri sağlamak için genelleştirilmiştir. Bir okuyucu-yazıcı kilidi edinmek, kilidin modunun belirtilmesini gerektirir: ya okuma ya da yazma erişimi. Bir işlem yalnızca paylaşılan verileri okumak istediğinde, okuma modunda okuyucu-yazıcı kilidini talep eder. Paylaşılan verileri değiştirmek isteyen bir işlem, yazma modunda kilidi talep etmelidir. Birden çok işlemin okuma modunda aynı anda bir okuyucu-yazar kilidi almasına izin verilir, ancak yazarlar için özel erişim gerektiğinden yalnızca bir işlem yazma için kilidi alabilir.



OKUYUCU YAZICI PROBLEMİ

Okuyucu-yazıcı kilitleri en çok aşağıdaki durumlarda kullanışlıdır:

• Hangi işlemlerin yalnızca paylaşılan verileri okuduğunu ve hangi işlemlerin yalnızca paylaşılan verileri yazdığını belirlemenin kolay olduğu uygulamalarda.

• Yazardan çok okuyucusu olan uygulamalarda. Bunun nedeni, okuyucu-yazar kilitlerinin oluşturulması için genellikle semaforlardan veya karşılıklı dışlama kilitlerinden daha fazla ek yük gerektirmesidir. Birden çok okuyucuya izin vermenin artan eşzamanlılığı, okuyucu-yazar kilidinin ayarlanmasıyla ilgili ek yükü telafi eder.