Prosesler Arası Etkileşim Semafor Yapısı ve Klasik Problemler

SEMAFOR YAPISI

- İncelenen çözümlerin zayıf yönleri:
  1. Proses sayısı arttıkça çözüm karmaşıklaşır
  2. Meşgul bekleme gerektirirler
  3. Donanıma bağımlılık

- SEMAFOR YAPISI

1. Meşgul bekleme içermeyen çözüm
2. Proses sayısından bağımsız
3. İşletim sistemi desteği gerektirir

SEMAFOR YAPISI

- Semafor işletim sistemi düzeyinde yaratılan, bir tamsayı ve bir işaretçi alanına sahip bir kayıt yapısında değişkendir.

- İşaretçi bir proses kuyruğuna işaretçidir, başangıçta null.
- Semafor değişkenine erişim ve değerinde yapılacak her hangi bir değişiklik ancak işletim sistemi tarafından, kesilemez şekilde gerçeklenen, iki özel fonksiyon ile mümkündür.
- Ayrıca, semafor yaratan ve ilk değerini atayan bir fonksiyon vardır.

SEMAFOR FONKSİYONLARI: P ve S

- P (S: semafor)
  if S>0 then S:=S-1
  else (prosesi bloke et);
- Eğer semafor değeri pozitif ise, değerini bir eksilt ve geri dön; aksi taktirde bu fonksiyonu yürütmüş olan prosesi bloke et ve S semaforu üzerinde bir bekleme kuyruğuna ekle
- V (S: semafor)
  if (S üzerinde bekleyen proses var)
  then (kuyruktan bir prosesi hazır kuyruğuna gönder)
  else S:=S+1;

SEMAFOR İLE KARŞILIKLI DIŞLAMA

- Kritik bölüme girmeden P(S), çıktıktan sonra V(S) yürütülür.
- Semaforun ilk değeri **"1"** olarak semafor yaratılmalıdır.

mx_begin: P(S);
mx_end: V(S);

```
Semaphore * S
/* ilk değeri 1*/

Begin
.....
P(s);
kritik bölüm;
V(s);
.....;
End;
```

SEMAFOR İLE SENKRONİZASYON

- SENKRONİZASYON:
- Senkronizasyon, bir olayı gerçekleştiren bir P1 prosesi ile çalışmasının belirli bir "s" noktasında, işleyişinin devamı bu olayın gerçekleşmesine bağlı olan bir diğer P2 prosesi arasındaki etkileşimdir.
- P2 prosesi "s" noktasına geldiğinde, söz konusu olay gerçekleşmiş ise, çalışma kesilmeden devam eder.
- Ancak, yürütme "s" noktasına ulaştığından henüz olay gerçekleşmemiş ise, P2 prosesinin çalışması kesilir, proses askıya alınır ve olay gerçekleşene kadar bekletilir.

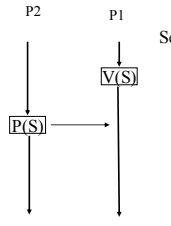
SEMAFOR İLE SENKRONİZASYON

- Senkronizasyon için kullanılacak olan S semaforu ilk değeri **"0"** olarak yaratılmalıdır.
- P1 prosesi olayı gerçekleştirdikten sonra V(S) işlemini yürütür.
- İlerlemesi olayın gerçekleşmesine bağlı olan P2 prosesi yürütmenin "s" noktasına geldiğinde, ilerlemeden önce P(S) işlemini yürütür.
- Semaforun o anda taşıdığı değere göre, P2
  i. takılmadan ilerleyecek (olay once gerçekleşmiş),veya
  ii. askıya alınıp bekletilecektir (olay henüz gerçekleşmemiş).

SEMAFOR İLE SENKRONİZASYON

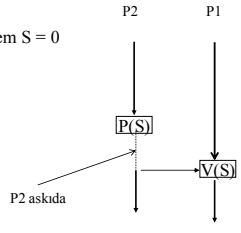
- İki durum söz konusu olabilir:

i: olay senkronizasyon noktasından **önce** gerçekleşir



Sem S = 0

ii: olay senkronizasyon noktasından **sonra** gerçekleşir

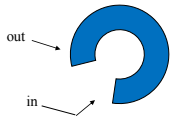


Sem S = 0

- Karşılıklı dışlama ve senkronizasyon gerektiren klasik problemler:
  - Üretici – tüketici problemi
  - Okuyucu – yazıcı problemi
  - Spagetti yiyen düşünürler problemi
  - Uyuyan berber problemi

ÜRETİCİ-TÜKETİCİ PROBLEMİ

- Farklı hızlarda çalışan üretici ve tüketici prosesler bir ortak alan üzerinden verilerini paylaşırlar.
- Üretici prosesler ortak alana ürettikleri verileri yazarlar, tüketici prosesler ise bu verileri okur ve işlerler.



N elemanlı çevrel kuyruk

```
int    in, out;
Item   buffer[n];
int    counter;
```

Üretici – Tüketici Problemi

- Dışlamalı erişim:
  - ortak tampon alanına erişim karşılıklı dışlama koşullarında gerçekleşmeli
- Senkronizasyon:
  - tampon boşsa tüketici beklemeli
  - tampon doluysa üretici beklemeli

ÜRETİCİ-TÜKETİCİ PROBLEMİ

- Problemler:
  i. Üretici yeni bir veri eklemek istediği zaman alanın dolu olması
  ii. Tüketici bir veri çekmek istediği zaman alanı boş bulması
- Her iki durumda da prosesler uygun koşullar oluşana kadar bloke edileceklerdir (meşgul bekleme çözümü kullanılmayacak)→senkronizasyon problemi
- Ortak alana erişim (kuyruk işlemleri, kuyruk işaretçilerinin güncellenme adımları) karşılıklı dışlama koşullarında gerçekleşmelidir.

İKİLİ ve SAYMA SEMAFORLARI

- Semaforları iki sınıfa ayırabiliriz:
  1. İkili semafor: değeri 0 veya 1 olabilen semafor
  2. Sayma semaforu: değeri "n" olabilen semafor

Sayma Semaforu: Kaynakların proseslere atanması için kullanılır.

- Semaforun ilk değeri ilk anda var olan kaynak sayısına eşitlenir.
- Kaynak elde etmek isteyen proses bir P işlemi yürütür (kaynak sayısını bir azaltır).
- Kaynaklar tükendiği zaman (sem değeri sıfır olmuştur), proses P işleminde askıya alınır.
- Kaynak kullanımı sona eren proses bir V işlemi ile bekleyen prosese kaynağını verir, bekleyen yoksa serbest kaynak sayısını (semafor değeri) bir artırır.

ÜRETİCİ-TÜKETİCİ PROBLEMİ

```
//N elemanlı tampon- tek üretici ve tek tüketici proses
typeT buf[n];
int out=0; in=0; // kuyruk işaretçileri
sem boş=n, dolu=0;

process Üretici {
while (true) {
...
// veri üret..
P(boş);
buf[in]=veri;in=(in+1)%n;
V (dolu);
}}

process Tüketici {
while (true) {
// tampondan veri çek
P(dolu);
veri=buf[out];out=(out+1)%n;
V(boş);
...
}}
```

ÜRETİCİ-TÜKETİCİ PROBLEMİ

```
//N elemanlı tampon - çok sayıda üretici ve tüketici
  prosesler
   typeT buf[n];
   int out=0; in=0;
   sem boş=n, dolu=0;
   sem dışla_U = 1, dışla_T=1;

   process Üretici [i=1 to M] {
   while (true) {
   ...
   // veri üret…
   P(boş);
    P(dışla_U);
    buf[in]=veri; in=(in+1)%n
    V(dışla_U);
   V (dolu);
   }}

   process Tüketici [j=1 to N] {
   while (true) {
   // tampondan veri çek
   P(dolu);
    P(dışla_T);
    veri=buf[out];out=(out+1)%n;
    V(dışla_T);
   V(boş);
   ...
   }}
```

OKUYUCU-YAZICI PROBLEMİ

- Çok sayıda okuyucu ve yazıcı proses ortak bir veri tabanına erişmektedirler.
- Birden fazla sayıda okuyucu proses, aynı anda, veri tabanını okuma işlemi için kullanabilir.
- Bir anda, sadece bir tane yazıcı prosese veri tabanını güncelleme izni verilebilir.
- SONUÇ: okuyucular paralel erişebilirler, ancak yazıcılar için dışlamalı erişim gereklidir.

OKUYUCU-YAZICI PROBLEMİ

```
sem okur_dışla=1; // okur_sayısı'na dışlamalı erişim
Sem  VT_dışla=1; // Veri tabanına dışlamalı erişim
int okur_sayısı=0;  // bir anda aktif olan okuyucu sayısı
```

Proses okuyucu:

```
{while (true) do {
  P(okur_dışla);
    okur_sayısı:= okur_sayısı + 1;
    if (okur_sayısı == 1) P(VT_dışla); //ilk okuyucu ??
  V(okur_dışla);

  .... Okuma işlemleri

  P(okur_dışla);
    okur_sayısı = okur_sayısı - 1;
    if (okur_sayısı == 0) V(VT_dışla); //son okuyucu ??
  V(okur_dışla); }}
```

OKUYUCU-YAZICI PROBLEMİ

```
sem okur_dışla=1; // okur_sayısı'na dışlamalı erişim
Sem  VT_dışla=1; // Veri tabanına dışlamalı erişim
int okur_sayısı=0;  // bir anda aktif olan okuyucu sayısı
```

Proses yazıcı:

```
{while (true) do {
  P(VT_dışla);

  .... Yazma işlemleri

  V(VT_dışla);}}
```

Okuyucu – Yazıcı Problemi

- Bu çözümde okuyucuların yazıcılara üstünlüğü vardır.
- Bir okuyucu sisteme girip de terk etmez ise, yazıcılar için sonsuz erteleme (indefinite postponement) söz konusu olur.
- Dengeli bir çözüm aranmalı
- Erişim sırası için kurallar uygulanmalı:
  - okuyucuların işinin bitmesini bekleyen yazıcı varsa yeni okuyucuya giriş izni verme.
  - yazıcının işinin bitmesini bekleyen okuyucuya yeni yazıcılara oranla öncelik tanı.

SPAGETTİ YİYEN DÜŞÜNÜRLER (DINING PHILOSOPHERS)

- N adet düşünür zamanlarını spagetti yiyip, düşünerek geçirirler.
- Yuvarlak bir masada N adet tabak ve N adet çatal bulunmaktadır.
- Düşünürün yemek için iki çatala (solundaki ve sağındaki) gereksinimi vardır.

PROBLEM: Ortak kullandıkları kaynakları ölümcül kilitlenme olmadan paylaşmak.

SPAGETTİ YİYEN DÜŞÜNÜRLER

- Ölümcül Kilitlenme durumu: Çatal elde etme isteklerinin hiç bir zaman kabul edilmemesi (sonsuz erteleme durumu-açlıktan ölüme neden olabilir☹)

- ÇÖZÜM: en yüksek paralelliğe izin veren bir algoritma geliştirilecektir.
- Dikkat: yan yana iki düşünür aynı anda yiyemezler!!!

SPAGETTİ YİYEN DÜŞÜNÜRLER

ÇÖZÜM 1: (????)

```
proses philosopher(i) {
while(1) {
düşün();
*çatal_al(i);          //sol çatalı al
 çatal_al((i+1)%N);    //sağ çatalı al
 …….  // yeme işlemi
 çatal_bırak(i);       //sol çatalı bırak
 çatal_bırak((i+1)%N); //sağ çatalı bırak
  }
}
```

Problem nedir??-KİLİTLENME

- Tüm prosesler "*" deyiminden sonra kesilirlerse, kilitlenme gerçekleşir.
- Her düşünür, sol elinde bir çatal, yanındakinin diğer çatalı bırakması için sonsuz beklemeye girer.

SPAGETTİ YİYEN DÜŞÜNÜRLER

ÇÖZÜM 2: "`düşün()`" deyiminden sonraki bölümün karşılıklı dışlama koşullarında gerçeklenmesi.

```
sem dışla=1;
proses philosopher(i) {
while(1) {
düşün();
P(dışla);
  çatal_al(i);          //sol çatalı al
  çatal_al((i+1)%N);    //sağ çatalı al
  …….  // yeme işlemi
  çatal_bırak(i);       //sol çatalı bırak
  çatal_bırak((i+1)%N); //sağ çatalı bırak
V(dışla);
   }
}
```

Problem nedir??-Kilitlenmeyi önler, ancak bir anda sadece bir düşünür yiyebilir→ paralelliğe izin vermez .

SPAGETTİ YİYEN DÜŞÜNÜRLER

Tanım ve bildirimler:

```
#define N 5 // düşünür sayısı
#define SAĞ(i) (((i)+1) %N)
#define SOL(i) (((i)==N) ? 0 : (i)+1)

typedef enum { DÜŞÜNÜYOR, AÇ, YİYOR} D_durumu;
D_durumu  durum[N];
semafor dışla=1;
semafor bekle[N]=0;
```

SPAGETTİ YİYEN DÜŞÜNÜRLER

```
proses düşünür(i) {
  while (true) {
    düşün();
    çatal_al(i);
    --- yemek ye ---
    çatal_bırak(i);
  }
}

çatal_al(i) {
  p(dışla);
   durum[i]=AÇ;
   dene(i);
  v(dışla);
  p(bekle[i]);
}
```

```
çatal_bırak(i) {
  sol=(i+4) % 5;
  sağ=(i+1) % 5;
  p(dışla);
   durum[i]=DÜŞÜNÜYOR;
   dene(sol);
   dene(sağ);
  v(dışla);
}

dene(i) {
  sol=(i+4) % 5;
  sağ=(i+1) % 5;
  if ((durum[i]=AÇ) ∧
      (durum[sol]≠YİYOR) ∧
      (durum[sağ]≠YİYOR))
     {
       durum[i]=YİYOR;
       v(bekle[i]);
     }
  }
```

SPAGETTİ YİYEN DÜŞÜNÜRLER

```
proses düşünür(int i;) {
while(true) {
   düşün();
   çatal_al(i);
   ye();
   çatal_bırak(i);
}
```

SPAGETTİ YİYEN DÜŞÜNÜRLER

```
void çatal_al(int i) {
 P(dışla);        // kritik bölüme gir
  durum[i] = AÇ; // yeme isteğini bildir
  dene(i);        // her iki çatalı almaya çalış
 V(dışla);
 P(bekle[i]);    // çatallar elde edilmediyse bekle
}
```

SPAGETTİ YİYEN DÜŞÜNÜRLER

```
void dene(int i) {
if ( durum[i] == AÇ            // ben aç isem ve
    && durum[SOL(i)] != YİYOR  // her iki komşum
    && durum[SAĞ(i)] != YİYOR ) // yemiyor ise
then
    { durum[i] = YİYOR; //durumu  değiştir
      V(bekle[i]);}
}
```

DİKKAT: eğer sağ ve sol çatallar serbest ise, kendi semaforu üzerinde V işlemi yürütür ve semafor değeri üzerinde bekleyen olmadığı için bir artar. Çatalları elde edemediği taktirde (then deyimine girmez), semafor değeri değişmez, ilk tanımında olduğu gibi 0 kalır!!

SPAGETTİ YİYEN DÜŞÜNÜRLER

```
void çatal_bırak(int i) {
  P(dışla);
   durum[i]=DÜŞÜNÜYOR;
   dene(SOL(i));  // sol ve sağ komşular
   dene(SAĞ(i));     adına çatalları elde
                     etmeye çalış
  V(dışla);
}
```