DONANIM DESTEKLİ ÇÖZÜMLER VE SEMAFOR YAPISI

DONANIM DESTEKLİ KARŞILIKLI DIŞLAMA

1. Kesmelere izin verme:
   i. Basit çözüm
   ii. Kesme hizmetlerinde gecikmelere neden olur
   iii. Çok işlemcili sistemlerde sorunu çözmez
2. Atomik işlemler kullan
   i. Bir tek adımda belleği kontrol edip, içeriğini değiştiren işlemler. Bu işlemler kesilemez işlemlerdir.
   ii. Test-And-Set
   iii. Exchange
   iv. Compare-And-Swap

Test-And-Set

Test-And-Set: 1. işlemciye özel makina komutu
2. atomik işlem olarak gerçekleşir

```
bool TestAndSet (bool & var) {
   bool temp;
   temp = var;
   var = TRUE;
return temp;}
```

Test-And-Set ile karşılıklı dışlama

```
bool lock:=false;  //ilk değer false atanır

mx_begin:
      while (TestAndSet(lock)) do;

mx_end:
      lock = FALSE;
```

Exchange

Exchange (Swap)

```
void Exchange(bool & a, bool & b){
bool temp;
        temp = a;
        a = b;
        b = temp;
}
```

Exchange ile Karşılıklı dışlama

```
bool lock:=false; //ilk değer false atanır

mx_begin:
   dummy = TRUE;
   while (dummy) Exchange(lock, dummy);

mx_end;
   lock =false;
```

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 **"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

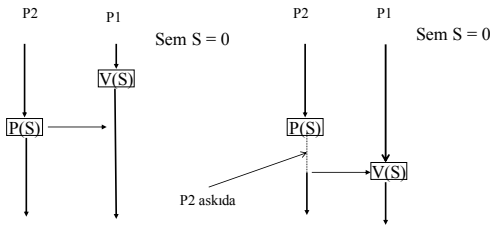
- Senkronizasyon için kullanılacak olan S semaforu ilk değeri **"0"** olarak yaratılmalıdır.
- P1 prosesi olayı gerçekleştirdikten sonra V(S) işlemini yürütür.
- İlerlemesi olayın gerçekleşmesine bağlı olan P2 prosesi yürütmenin "s" noktasına geldiğinde, ilerlemeden önce P(S) işlemini yürütür.
- Semaforun 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:

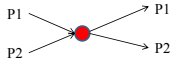
i: olay senkronizasyon noktasından **önce** gerçekleşir

ii: olay senkronizasyon noktasından **sonra** gerçekleşir



SEMAFOR İLE SENKRONİZASYON

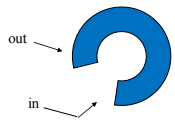
Buluşma: iki semafor kullanarak




```
sem sem1=0, sem2=0;
process P1 {
...
V(sem1); // buluşma noktasına geldiğini bildir
P(sem2); // diğer prosesi bekle
...
}
process P2 {
...
V(sem2); // buluşma noktasına geldiğini bildir
P(sem1); // diğer prosesi bekle
...
}
```

KLASİK SENKRONİZASYON PROBLEMLERİ

- 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;
```

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).
- Ortak alana erişim (kuyruk işlemleri, kuyruk işaretçilerinin güncellenme adımları) karşılıklı dışlama koşullarında gerçekleşmelidir.

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);}}
```

- Bu çözümde okuyucuların yazıcılara üstünlüğü vardır.
- Bir okuyucu sisteme girip de terk etmez ise, yazıcılar için sonsuz erteleme (indefinite postponement) söz konusu olur.
- ÇÖZÜM??
- Bekleyen yazıcı var ise, yeni okuyucuya izin verme.
  - Bekleyen yazıcı sayısını tut, dışlamalı erişim sağla