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

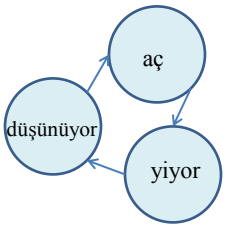
ÇÖZÜM 2: "`düşün()`" deyiminden sonraki bölümün karşılıklı dışlama koşullarında gerçeklenmesi.

```
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

ÇÖZÜM (Tannenbaum): Düşünürlerin içinde bulundukları durumları ve durumlar arasındaki geçiş koşullarını belirle.



- Her düşünürün içinde bulunduğu durumu göstermek üzere bir dizi kullan:

```
integer durum[N];
 // durum [i]=0---düşünüyor
 // durum [i]=1---aç, çatal bekliyor
 // durum [i]=2---yiyor
```

- Her düşünür için, yeme isteğinde bulunup çatalları elde edememesi halinde üzerinde bloke olacağı bir semafor:

```
semafor bekle[N]=0;
```

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

```
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
}

 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]);}
 }


 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);
 }
```

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);
}
```

MONİTÖR YAPISI

- Semafor işletim sistemi düzeyinde temel senkronizasyon yapısı, ancak kullanımı zor, dikkat gerektirir.
- Kritik bölümler kod içinde dağınık şekilde yer alabilirler→paralel kodun incelenmesi, hata ayıklama güçleşir.
- ÇÖZÜM: paylaşılan verileri işleyen kod parçalarını bir noktaya toplama. Sınıf yapısına benzer bir yapı altında, veriler üzerindeki tüm işlemler tek bir merkezi yapı tarafından gerçekleşsin. İşletim sisteminden destek alan, programlama dili düzeyinde bir yapı kullanım kolaylığı sağlayacaktır.

MONİTÖR YAPISI

- MONİTÖR YAPISI: *C.A.R. Hoare, CACM, vol 17, no. 10, Oct. 1974*
- 1971'de Dijkstra tarafından önerildi,
  1972-73 Hansen kavramı geliştirdi, ve
  1974'te Hoare son halini verdi.

- Monitor yapısı bir yüksek düzeyli dil tarafından desteklenir.
  - İlk öneri Concurrent Pascal için gerçekleştirilmiştir.
- Her monitör belirli bir görev için tanımlanır.
- Ortak değişkenlerin çok sayıda proses arasında etkin ve güvenli bir şekilde paylaşılmasına olanak sağlar.

MONİTÖR YAPISI

- Her monitör'ün kendine ait yerel verileri vardır (paylaşılan ortak değişkenler).
- Bu verilere sadece monitör prosedürleri (entry) aracılığıyla erişilebilir.
- Proseslerin bu verilere erişmek için monitör prosedürlerini (entry) çağırmaları gerekir.
- Monitör değişkenleri tüm yürütme boyunca değerlerini kaybetmezler.
- Entry'ler yerel değişkenlere sahip olabilirler; çağrı sonlanınca yerel değişkenler değerlerini kaybederler.

Monitör Sentaksı

Monitör bildirimi:

```
monitor monitor-adı
{       ……..//      ortak değişken bildirimleri

        entry P1( … ) {
          ……..//  yerel değişken bildirimleri        }
        entry P2( … ) {
                …
        }
                .
                .
        entry Pn( … ) {
                …
        }
        {
                // başlangıç kodu, ortak
                değişkenlere ilk değerleri atanır
        }
}
```

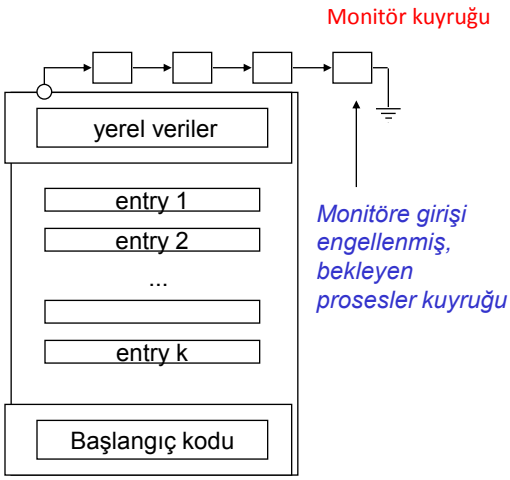
Entry çağrısı:

```
call monitor_adı.entry_adı (argüman listesi);
```

MONİTÖR YAPISI

- *** Yürütmenin her hangi bir anında sadece **bir** prosesin monitör içinde aktif olmasına (bir entry kodu yürütüyor olmasına) izin verilir. → entry kodları birbirlerini dışlayarak yürütülür.
- Bir proses monitör içinde aktif iken, bir diğer prosesin bir entry çağrısı yapması halinde, çağrıyı yapan proses bloke edilir ve ***monitöre girmek üzere bekleyen prosesler kuyruğu'na (monitör kuyruğu)*** eklenir.
- Bir proses monitörü terk edince, monitör kuyruğunda bekleyen bir prosesin (var ise) ilerlemesine izin verilir.

MONİTÖRÜN İÇ YAPISI



Örnek: Olay gözleme/rapor etme

```
monitor gözle {
  integer     sayaç;
  entry  void say() {
      sayaç:=sayaç+1;
  }
  entry void yaz(){
      print (sayaç);
     sayaç:=0;
  }
  { sayaç:=0;}
}
```

```
Process gözleyici;
{… // olay gözle
  call gözle.say();
}

Process raportör;
{… // bekle
  call gözle.yaz();
}
```

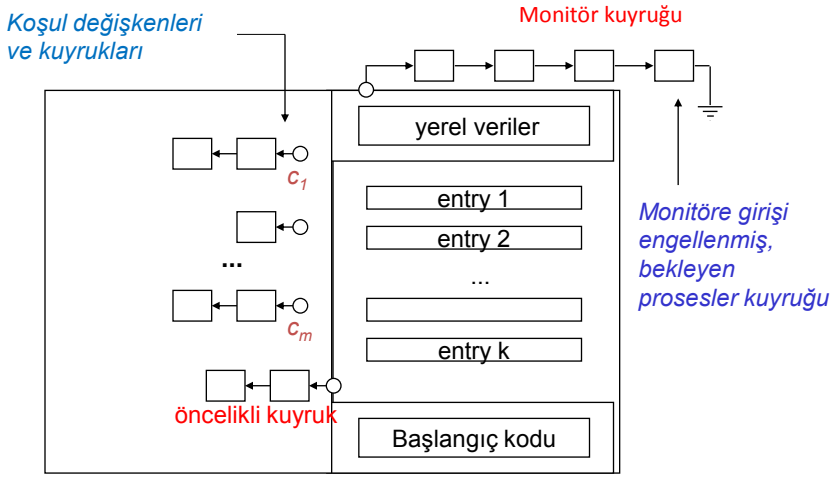
KOŞUL DEĞİŞKENLERİ

- Monitör prosedürlerinin dışlamalı yürütülmesi garanti edilmiştir, ancak diğer senkronizasyon problemlerinin çözümü programcıya bırakılmıştır.
- Örnek problem: monitör bir kaynağın kullanımını paylaştırır.
  - Kaynağı elde etmek için monitör çağrısı yürüten proses, kaynağın hazır olmaması halinde bloke olacaktır. Ancak, monitör içinde kaldığı için diğer proseslerin monitöre girişine engel olur, kilitlenme oluşur.
- Bu gibi durumlarda, proseslerin bloke olmadan önce monitörü terk etmelerini sağlayan özel bir yapı tanımlanmıştır: Koşul değişkenleri (condition variables)
- Böylece, bloke olan proses diğer proseslerin monitöre erişimlerine engel olmadan, monitör dışında, koşulların elverişli hale gelmesini bekleyebilir.

KOŞUL DEĞİŞKENLERİ

- Koşul değişkeni: "condition" tipinden tanımlanmış değişken.
  - c:condition;
- Bu değişkenlere iki özel işlem uygulanır: cwait ve csignal
- cwait(c): çağrıyı yapan prosesi askıya al ve koşul değişkenine ait bir bekleme kuyruğuna yerleştir. Artık monitör diğer proses çağrılarına açıktır.
- csignal(c): koşul değişkeni kuyruğunda bekleyen proses yok ise bir işlem yürütmeden geri dön; var ise, onu canlandır. Bu durumda csignal yürüten proses monitörden çıkarılır ve bir öncelikli bekleme kuyruğuna alınır. Canlandırılan proses ise derhal monitöre girer ve son yürütmüş olduğu "cwait" çağrısını izleyen adımdan devam eder. Monitör boşaldığı zaman öncelikli kuyrukta yer alan proseslere öncelik tanınır.

MONİTÖRÜN İÇ YAPISI



Örnek: Tek kullanıcılı kaynak paylaşımı

```
monitor kaynak {

  var bool    meşgul;
  condition   hazır;

  entry  void al() {
      if (meşgul) cwait(hazır);
      mesgul:= TRUE;
  }

  entry void ver() {
      meşgul = FALSE;
      csignal(hazır);
  }

  {mesgul:=FALSE;}
}
```

```
Proses kullanıcı
{..
  call kaynak.al();
   ……. // kaynağı kullan
  call kaynak.ver();
 ..
}
```

Örnek: İkili Semafor

```
monitor semafor {

  var bool    kilitli;
  condition   durum;

  entry  void P() {
      if (kilitli) cwait(durum);
      kilitli:= TRUE;
  }

  entry void V() {
      kilitli = FALSE;
      csignal(durum);
  }

  {kilitli:=FALSE;}
}
```

İkili semafor tek kullanıcılı bir kaynak olarak gerçeklenir.

Monitör Yapısının Gerçeklenmesi

- Monitör yapısı semafor kullanarak gerçeklenir.
- Monitör prosedürlerinin dışlamalı yürütülmesi (monitör içinde aktif bir prosesin olması) için bir ikili semafor "dışla".

  semafor dışla = 1;
- csignal ile monitörü terk eden prosesin monitör dışında üzerinde bekleyeceği bir ikili semafor "önce_gir".

  semafor önce_gir = 0;
- Monitöre öncelikli olarak girmek üzere bekleyen proses sayısını tutan tamsayı değişkeni "önce_sayısı".

  int önce_sayısı=0;

Monitör Yapısının Gerçeklenmesi

- Derleyici, her entry kodunun başına ve sonuna şu deyimleri ekler

```
Entry E();
{   P(dışla);
    ...
    ...   // E'e ait deyimler
    ...
    if (önce_sayısı>0)
          V(önce_gir);
    else
          V(dışla);
};
```

Monitör Yapısının Gerçeklenmesi

- Monitör içinde tanımlı her koşul değişkeni için ilk değeri 0 olan ayrı bir semafor tanımlanır.

  semafor koşul_sem = 0;

- Koşul değişkeni üzerinde bekleyen proses sayısı için bir sayaç.

  int koşul_sayısı=0;

```
cwait(koşul_x) {
   koşul_x_sayısı ++;
   if (önce_sayısı >0)
       V(önce_gir);
   else
       V(dışla);
   P(koşul_x.sem);
    koşul_x_sayısı --;
}
```

```
x.signal(koşul_x) {
    if (koşul_x_sayısı>0){
      önce_sayısı++;
      V(koşul_x.sem);
      P(önce_gir);
      önce_sayısı --;
    }
}
```

Örnek: Üretici-Tüketici problemi

```
monitor tampon {
  int B[1..n];
  int gelen, çıkan, sayaç;
  condition dolu, boş;

  entry ekle(x:int)
  {
    if (sayaç==n)
      then cwait(boş);
    B[gelen]= x;
    gelen=(gelen+1)mod n;
    sayaç++;
    csignal(dolu);
  }
```

```
entry çıkar(x:int)
  {
    if (sayaç==0)
      then cwait(dolu);
    x=B[çıkan];
    çıkan=(çıkan+1)mod n;
    sayaç--;
    csignal(boş);
  }

  {
   gelen=0;
   çıkan=0;
   sayaç=0;

  }
 }
```

Örnek: Üretici-Tüketici problemi

```
proses Üretici;
{
  while true do
  {
    .. veri üret

    tampon.ekle(x);
  };
};
```

```
proses Tüketici;
{
  while true do
  {
    tampon.çıkar(x);
    .. veriyi kullan
  };
};
```

Örnek: Spagetti Yiyen Düşünürler Problemi

```
monitor SPD {
      condition bekle [0..4];
      typedef durumlar =(düşün,aç,yiyor);
      durumlar durum[0..4] = düşün(5);  // hepsi düşünüyor

      entry çatal_al(i:int);
      {
         durum [i] = aç;
         dene (i);
         if ( durum [i] != yiyor ) cwait ( bekle [i] );
      }
      entry çatal_bırak (i:int);
      {
         durum [i] = düşün;
         dene ( (i+4)%5 );  // sağ komşu için dene
         dene ( (i+1)%5 );  // sol komşu için dene
      }
```

Örnek: Spagetti Yiyen Düşünürler Problemi

```
procedure dene (int k) {
   if ( durum [ (k+4)%5 ] != yiyor
          && durum [k]==aç
          && durum [ (k+1)%5 ] != yiyor )
      { durum [k] = yiyor;
        signal ( bekle[k] );
      }
}}
```

```
Process düşünür (i);
{
  while true do
  {
    SPD.çatal_al (i);
     …… // spagetti ye
    SPD.çatal_bırak (i);
  };
};
```

Örnek: Uyuyan Berber Problemi

- Bir berber, bir berber koltuğu ve n adet müşteri koltuğu mevcuttur.
- Berber bir müşteri gelene kadar uyur.
- İlk müşteri berberi uyandırır. Daha sonra gelen müşteriler boş koltuk varsa oturup beklerler, yoksa giderler.
- Berberi ve müşterileri yarış durumuna neden olmayacak şekilde programlayın.

Örnek: Uyuyan Berber Problemi

```
Monitor uyuyan_berber {
       condition  müşteri_bekle, berber_bekle;
       int bekleyen = 0;  // berberi bekleyen müşteri sayısı

       entry berber {
            if (bekleyen==0)then
               cwait(müşteri_bekle); // müşteri yoksa bekle
            saç_kes(); …… berber müşterinin saçını keser
            bekleyen = bekleyen -1;
            csignal(berber_bekle);  // bekleyen müşteriyi uyar

       }
       entry saç_kestir {
          if (bekleyen < "koltuk sayısı") { // boş koltuk varsa
             bekleyen = bekleyen +1;
             csignal(müşteri_bekle);
             cwait(berber_bekle);
             …… müşterinin saçı kesilir
          }
       }
    }
```