# PARALEL PROGRAMLAMA

ARDIŞIL PROGRAM:

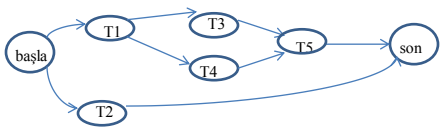
Birbirini izleyen sırada yürütülen deyimlerden oluşan program. Bu tür programların her birine bir proses diyebiliriz.

PARALEL PROGRAM:

Belirli bir problemin çözümü için, paralel (lojik olarak) yürütülmekte olan ardışıl prosesler topluluğu. Her prosesin kendine ait bir kontrol akışı vardır.

## PARALEL YAPILAR

- Birden fazla işlemin veya makinanın aynı anda aktif olmalarını gerektiren durumlar paralel bir yapı gösterirler.
- ÖRNEK 1: bir projenin iş akış planı

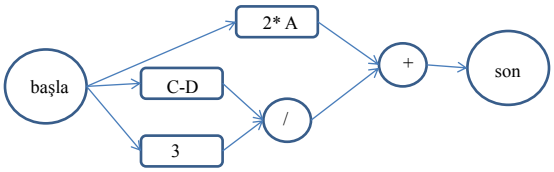



- T1 ve T2 başlangıçta paralel yürütülebilecek işlemlerdir.
- T3 ve T4 ancak T1 tamamlandıktan sonra başlayabilir.
- Projenin sonra ermesi için T2 ve T5'in bitmiş olması gerekir.

## PARALEL YAPILAR

- ÖRNEK 2: Aritmetik ifadelerin değerlendirilmesi

(2 * A ) + ((C-D) / 3)




İfade ardışıl olarak değerlendirilebileceği gibi, işlemler arasındaki öncelik sıralamasını da dikkate alarak, bazı işlemlerin diğerleri ile paralel yürütülebilmesinin mümkün olduğu görülür.

## PARALEL YAPILAR

- ÖRNEK 3: gerçek zamanda sistem kontrolü

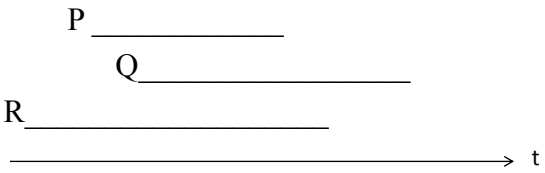
Bir ürünün üretim aşamaları denetlenecek ise çok sayıda dış etken sürekli izlenmek durumundadır. Uygun bir yaklaşım her etken için ayrı bir program biriminin etkin kılınması ile paralel olarak denetlenmesi olacaktır.

- ÖRNEK 4: benzetim sistemleri
- ÖRNEK 5: bilgisayar işletim sistemleri

Sonuç: Çözüm bulunmaya çalışılan birçok problemin doğal olarak bir paralel yapı içerdikleri görülür.

## PROSES

- Proses: Birbirini izleyen sırada yürütülen işlemler dizisi.
- Paralel proses: Yürütülme süreleri zaman içinde çakışan prosesler. Örnek: P,Q,R: üç proses ise,




- Paralel program: İki veya daha fazla paralel prosesten oluşan program. Paralel prosesler değişik donanımlar üzerinde yürütülebilirler: tek işlemci veya çok işlemcili donanımlar.

## PROSES

- Tek işlemcili donanım: zaman paylaşımı ile çok sayıda proses yürütülebilir - görünürde paralel çalışma söz konusudur.
  - Proseslerin yürütülme sıraları ve hızları hakkında fikir yürütülemez. Bu nedenle, proseslerin yürütülme sürelerini dikkate alan kod yazılmamalıdır.
- Çok işlemcili donanım:Gerçek anlamda paralellik vardır

i. Çoklu işleme: Mimari bir *ortak belleği* paylaşan çok sayıdaki işlemciden oluşur. Her işlemci bir veya daha fazla sayıda prosesi zaman paylaşımlı yürütür.

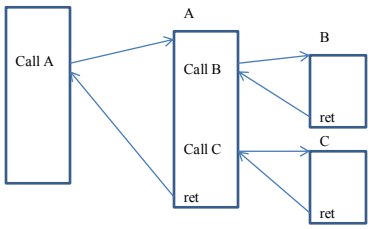
ii. Dağıtılmış işleme: Çok sayıda işlemci bir ağ üzerinden birbirine bağlıdır (ortak bellek yoktur).

## PROSES ETKİLEŞİMİ

- Ayrık proses: farklı veri kümeleri üzerinde, birbirlerinden bağımsız işlemler yürüten prosesler.
- Ancak çoğunlukla, ortak bir hedef için çalışan prosesler arasında zaman zaman etkileşimde bulunma gereği doğar.
  - Elde edilen sonuçların birbirlerine aktarılması: uygun bir haberleşme mekanizması var olmalıdır (ortak değişkenler veya mesaj aktarımı)
  - Senkronizasyonun sağlanması: bir proses tarafından yürütülen işlemlerden bir bölümünün gerçekleştirilmesi, bir başka prosesin denetimindeki bazı koşullara bağlı olabilir. Senkronizasyonu sağlayan yapılar var olmalıdır.

## PARALEL YÜRÜTMENİN GÖSTERİLİMİ

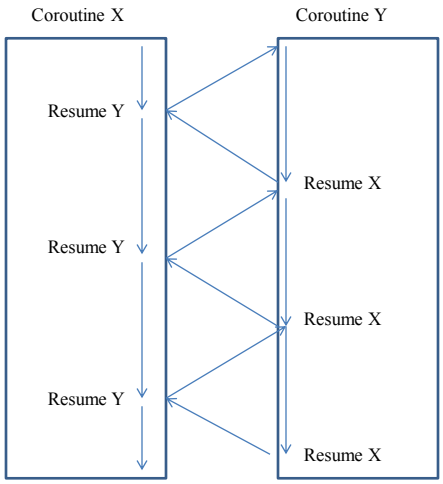
- COROUTINE (CONWAY, 1963)
  - •Altprogramlara benzer, ancak aralarında geçiş *simetriktir*.
  - •Altprogramlar arasında *hiyerarşik* bir yapı vardır, her çağrıda kontrol altprogramın başlangıç noktasına geçer, return ile karşılaşılınca çağrının yapıldığı noktaya geri döner.



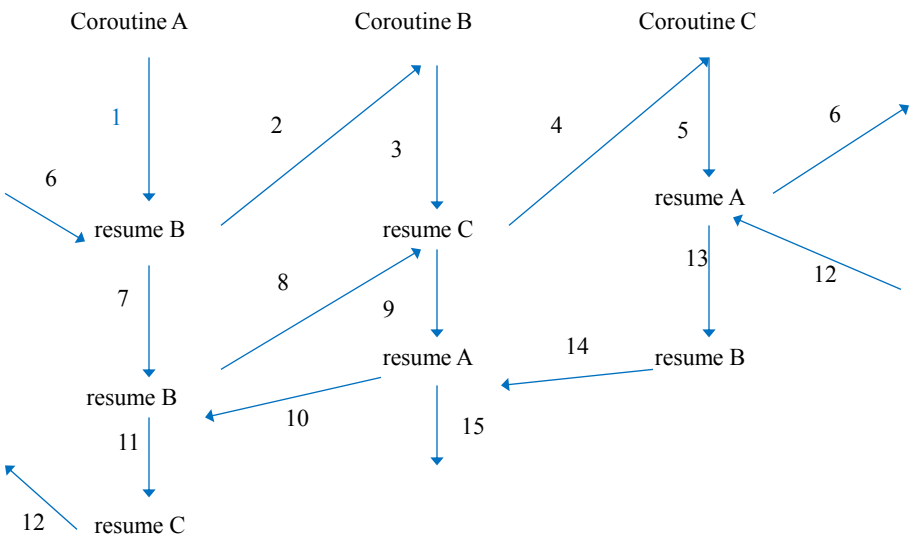
## COROUTINE

- Coroutine'lerde *resume* deyimi kontrolü bir başka coroutine'e geçirir. Geçiş şöyle gerçekleşir.
  1. Kontrolün resume deyimini izleyen deyime geçmesini sağlayacak bilgileri saklanır.
  2. Çağrılan coroutine ilk kez canlanıyor ise, kontrol ilk yürütülebilir deyime geçer.
  3. Çağrılan coroutine daha önce aktif olmuş ise, bu kez kontrol yürütülmüş olan son resume deyimini izleyen deyime geçer.
- Coroutine'ler return değil, resume deyimleri ile kontrolü aralarında paylaşırlar. Resume etkisi, çalışmayı sonlandırmak değil, bir süre askıya almaktır.

## COROUTINE



## Coroutine Kontrol Akışı




## COROUTINE

- Coroutine için örnek problem:
- KAR bir karakter dizisidir. NEXT altprogramı, her çağrılışında, dizinin bir sonraki karakteri olan KAR[I] değerini bir C değişkeninde geri getirir. Bellekten tasarruf amacıyla, boşluklardan oluşan katarlar KAR dizisi içinde boşluk karakteri ve onu izleyen alana boşluk sayısı bilgisi eklenerek gösterilmiştir (örneğin 5 adet boşluk için " boşluk 5"). Next, bu durumu dikkate alıp, ana programa gereken sayıda boşluk karakteri göndermelidir.

# ALTPROGRAM İLE ÇÖZÜM

```
Subroutine NEXT;
        //Boşluk katarı mı üretiliyor?  KALAN değişkeni kaç adet boşluk üretileceğini gösterir
if  KALAN==0 then   //değil, bir sonraki karakteri bul
   begin
      I:=I+1
      if KAR[I] <> boşluk then  C:= KAR[I]   // C'i          anaprograma gönder
      else begin    //boşluk katarı var
                I:=I+1;
                KALAN:= KAR[I] – 1;  //KALAN'a boşluk sayısının bir eksiğini ata
                C:= boşluk;
           end;
   end;
else  // boşluk katarı işleniyor
  begin
     C:= boşluk;
     KALAN:=KALAN-1;
  end;
return;
```

# COROUTINE İLE ÇÖZÜM

- Coroutine ile çözüm:

```
Coroutine NEXT;
      // bu örnek sonsuz çevrim şeklinde yazılmıştır
  ……                                    MAIN:
  LOOP:   I:=I+1                        LOOP1: resume Next;
       if  KAR[I] <> boşluk then                ……   // C'i işle
       begin                                    goto LOOP1;
         C:= KAR[I] ;
          resume  MAIN;
       end;
```

## COROUTINE

```
            else
               begin
                  I:=I+1;
                  for J:=1 until KAR[I] do
                     begin   //boşluklar bitene kadar ana programa gidiş
                                 geliş bu çevrim içinde gerçekleşir
                        C:= boşluk;
                        resume  MAIN;
                      end;
               end;
               go to LOOP;
end  NEXT;
```

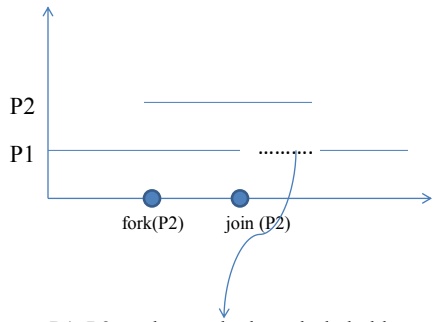
## FORK/JOIN

- FORK/JOIN deyimleri:
- *fork(x):* belirlediği programın (x) yürütülmeye başlanmasını sağlar. Ancak, fork yürüten program da çalışmasını sürdürür→ aynı anda etkin olan iki program ortaya çıkar
- *join (x):* fork ile çalışmaya başlayan programlar arasında senkronizasyon sağlar. Bu deyim, kendisini yürüten programı, join ile belirlenen program (x) sona erene kadar askıya alarak bekletir.

# FORK/JOIN

- Program P1;
  ……..
  fork (P2);
  …..
  …..
  join (P2);
  …..

Program P2;
…..
…..
…..
end;



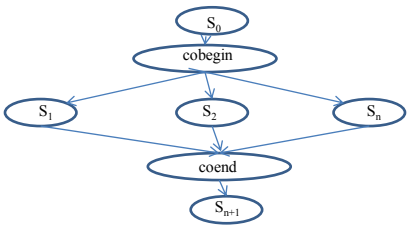

P1, P2 sonlanana kadar askıda bekler

# COBEGIN / COEND

- COBEGIN/COEND deyimi (Dijkstra):
- cobegin-coend, bu anahtar sözcükler arasında yer alan deyimlerin paralel yürütüleceğini belirtir.

- Örnek:

$S_0$;
cobegin
$S_1$;
$S_2$;
……
$S_n$;
coend;
$S_{n+1}$;

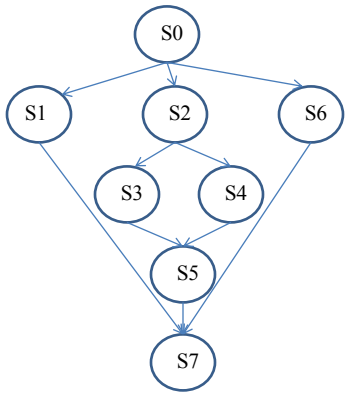



-Önce S0 çalışır ve sonlanır.
-S1, S2,…Sn paralel olarak yürütülür.
-S1, S2,…Sn deyimlerinin tümü sonlandıktan sonra, Sn+1 yürütülür

## COBEGIN / COEND

- Cobegin-coend deyimleri iç içe de kullanılabilir.

```
S0;
  cobegin
      S1;
      begin
        S2;
        cobegin
          S3; S4;
        coend;
        S5;
      end;
      S6;
  coend;
  S7;
```




## COBEGIN / COEND

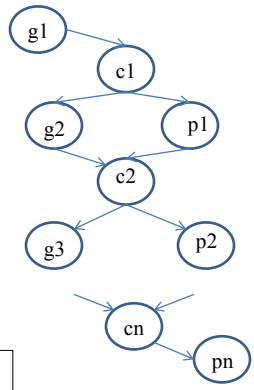
- Örnek: f ve g, T tipinden kayıtlara sahip olan iki dosyadır. Kayıtlar f dosyasından okunup, g dosyasına yazılacaktır.

```
procedure copy(var f,g: sequence of T);
var s,t:T; completed: boolean;

begin
   if not  empty(f) then
   begin
      completed:= false;
      get (s,f);
      repeat
       …….
```

## COBEGIN / COEND

```
repeat
   t:=s;
   cobegin
      put (t,g);
      if empty (f)
          then completed:=true
          else get (s,f);
   coend;
until completed;
end;
end;
```




g: get
p: put
c: t←s

## PROSES/TASK BİLDİRİMİ

PROSES/TASK:

Program kodu, paralel yürütülmesi öngörülen prosedürler topluluğu şeklinde ifade edilir. Her prosedür kendi içinde ardışıl olarak yürütülen deyimlerden oluşur. Bu prosedürlere PROSES/TASK adı verilir.

```
Process  P:
begin
 . . .
end;
```

İşletim sistemi düzeyinde bir yazılım veya derleyicinin ürettiği özel bir kod sayesinde proseslerin aynı anda çalışmaya başlamaları sağlanır.

## PROSESLERİN YARATILMASI

- Bazı dillerde tanımı yapılan proseslerin yalnızca bir örneği yaratılır ve canlandırılır. Yürütme sırasında proses sayısı sabit olduğu için statik bir yapı oluşur. (Distributed Process/Hansen)
- Bazı dillerde, bir proses bildiriminin birden fazla örneğini (instance) yaratmak mümkündür. Ancak bu işlem sadece yürütme başlamadan önce gerçekleştirilebilir→ proses sayısı yine sabit kalır. (Modula, Concurrent Pascal)
- Bazı diller ise, yürütme sırasında , gerek duyuldukça yeni proses/task yaratılmasına izin verir. Task tiplerinin tanımı yapılabilir ve aynı tipten çok sayıda task yaratılabilir. Bu durumda değişken sayıda proses/task ile dinamik bir yapı kurulabilir.