OCCAM
(I)

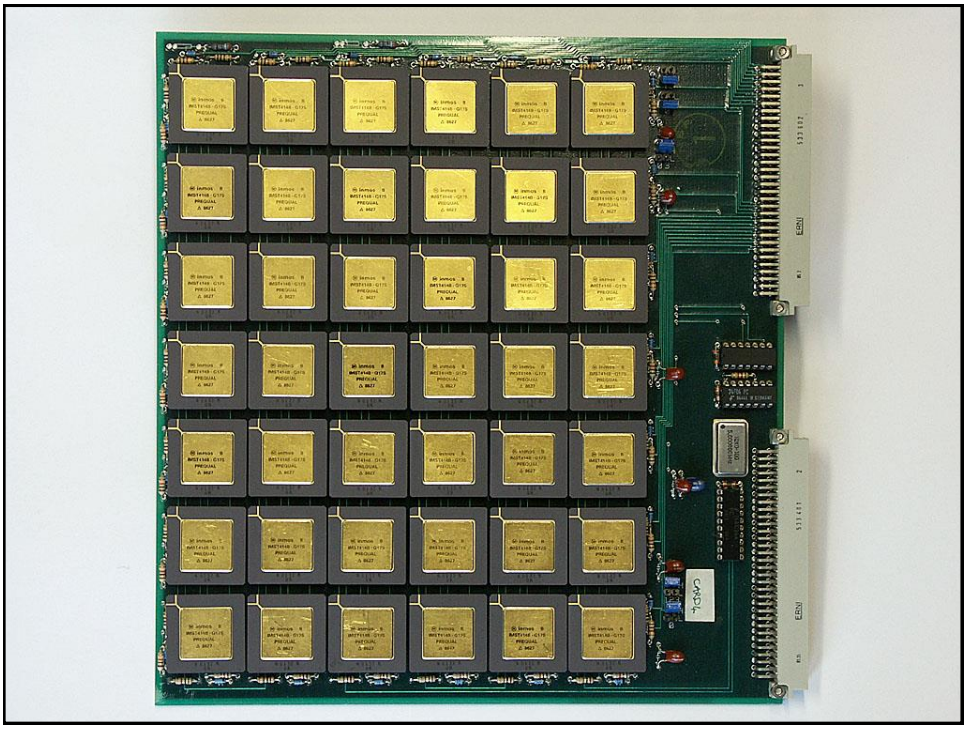
OCCAM

- William of Occam, 14yy düşünürü
- *"Entia non sunt multiplicanda sine necessitate*"
- "Entities are not to be multiplied beyond necessity."
- Gerek duyulmadıkça, herşeyi basit tutulmalıdır.

- Occam dili mümkün olan en küçük ve sade haliyle tasarlanmış, bu nedenle bu adı almıştır.

TRANSPUTER

- Occam dili INMOS tarafından geliştirilen işlemcinin (transputer) simgesel dilidir.
- Transputer paralel çalışmaya donanım düzeyinde destek sağlayan hızlı bir işlemci.
- Transputer kırmığı bileşenleri şunlardır:
  - Bir işlemci
  - Bellek
  - Co-processor
  - Dört adet yüksek hızlı, çift yönlü iletişim hattı
    - 1.8 Mbytes/sec (T8/T4/T2)
    - 10 Mbytes/sec (T9000)



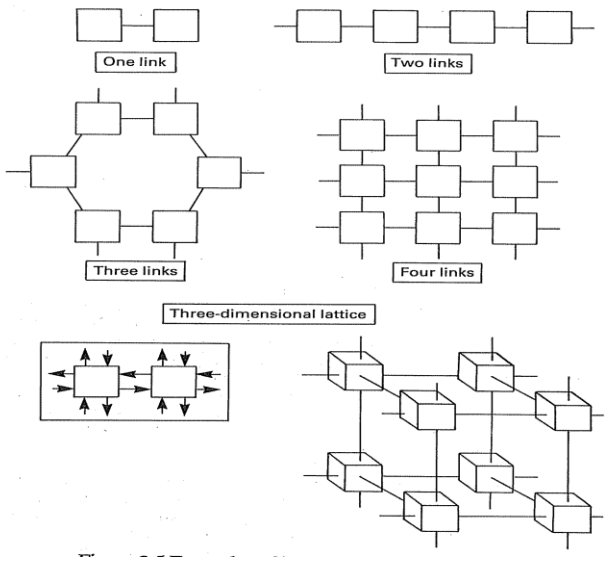
TRANSPUTER

- İşlemci: çok görevli çalışmayı (multi tasking) destekler. Paralel prosesler için iki farklı öncelik düzeyine sahip olabilirler.
- İletişim hatları: Transputer'lar birbirleriyle hızlı iletişim hatları (serial link) üzerinden haberleşirler.
- Yazılım düzeyinde, iletişim hatları kanal (channel) olarak tanımlanmıştır. Paralel çalışan prosesler kanallar üzerinden haberleşirler.

TRANSPUTER

- Bir transputer iletişim hatları üzerinden 4 farklı transputer'a çok basit bir donanımla bağlanabilir.
- Yüksek sayıda işlemciden oluşan konfigürasyonlar kolaylıkla oluşturulabilir (ring, mesh, hypercube, tree, vs.).
- Aralarında ortak bellek paylaşımı söz konusu değildir.
- Birbirlerine bağlanacak transputer sayısı üzerinde bir üst sınır olmadığı için, yüksek işlem gücü elde edilebilir.
- Bu tür işlem gücünden doğru yararlanabilmek için paralel çalışan algoritmalara gerek vardır.

ÖRNEK MİMARİLER



TRANSPUTER

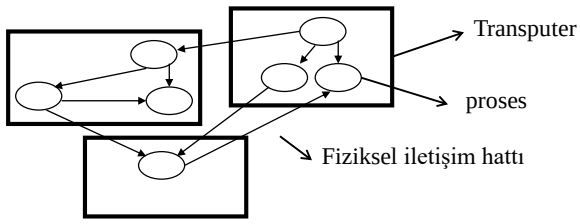
- Kullanıcılar, proseslerin konumundan bağımsız olarak (aynı işlemci veya farklı işlemciler) iletişim için aynı mesajlaşma arayüzünü kullanırlar.
- Prosesler farklı işlemcilerde yer alıyor iseler, mesajlar iletişim hatları üzerinden aktarılır. Eğer transputer'lar doğrudan birbirlerine bağlı değil iseler, sanal hatlar (virtual link) üzerinden aktarım gerçekleştirilir.
- Aynı işlemcide yer alan prosesler için, mesaj aktarımı bellek üzerinde oluşturulan lojik kanallar üzerinden gerçekleştirilir.

TRANSPUTER

- Transputer dilin gerçeklenmesini sağlayacak çekirdek yazılımına ait birçok fonksiyonu (bağlam değiştirme, iş sıralama, kuyruk işlemleri, vs.) donanım düzeyinde destekler.
- Kaynak bekleyen prosesler işlemci zamanı tüketmezler. Bağlam değişikliği 1 μsec kadar kısa sürede tamamlanır.
- Transputer'lar arasındaki iletişim hatları işlemci ile paralel çalışırlar ve işlemci müdahalesi olmadan her dört hat üzerinden veri aktarabilirler.

OCCAM PROSESLERİ

- Occam CSP diline dayalı paralel programlamayı destekleyen bir dildir.
- Bir paralel programın temelini birbirleriyle haberleşen prosesler ağı oluşturur.



- Prosesler var olan transputer işlemcilerine atanırlar, kanallar da iletişim hatları ile ilişkilendirilirler.

OCCAM PROSESLERİ

- Occam dilinde yürütülebilir **her deyim bir prosestir**.
- Çok sayıda proses (> 100,000 paralel proses) var olacağı için prosesler adlandırılmazlar.
- Bir proses diğer prosesleri içerebilir: prosesler arasında hiyerarşik bir yapı vardır.
- Occam programının kendisi de bir tek proses olarak kabul edilir.
- Her deyim (her proses) bir program satırına yerleşir.
- Prosesler ortak belleği paylaşmazlar, birbirleriyle mesaj aktarımı yolu ile haberleşirler.

OCCAM PROSESLERİ

- Occam prosesleri şu düzenlerde olabilirler:
  - İlkel prosesler (primitive processes)
  - Blok prosesler (block processes)
  - Kurucu prosesler (constructed processes)
  - Prosedür örnekleri (procedure instances)
- İlkel prosesler: bu tür prosesler **kesilemez** şekilde yürütülürler
  - Input
  - Output
  - Atama
  - Stop
  - Skip

PROSES İLETİŞİMİ

- Prosesler kanallar üzerinden veri aktararak haberleşirler.
- Mesajlaşma: buluşma (rendez-vous) şeklinde gerçekleşir
  - Senkron,
  - Tamponlama yok,
  - Bire-bir.
- Kanal (channel):
  - Tek yönlü veri aktarır.
  - Yalnızca iki proses tarafından kullanılabilir: kanala bilgi gönderen ve kanaldan bilgi okuyan prosesler.
  - İsimlendirme olmadığından, bir kanalın hangi iki proses arasında bağlantı kurduğu kullanımdan ortaya çıkar; hatalı durumları derleyici yakalar.

PROSES İLETİŞİMİ

- Bir kanal üzerinden iletişim iki temel eylemle gerçekleşir:
  - Input: kanaldan bir veri okur ve alınan değeri bir değişkene atar

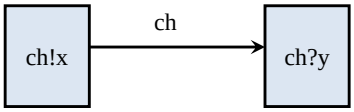
```
keyboard ? char
```

  - Sentaks: *input* = *channel* ? *variable*
  - Output: bir ifadenin değerini kanala gönderir

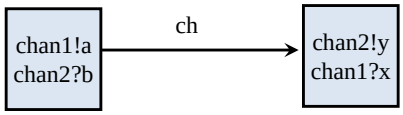
```
screen ! char
```

  - Sentaks: *output* = *channel* ! *expression*
- Input ve output birer ilkel prosestir.

PROSES İLETİŞİMİ:BULUŞMA



- output prosesi: “ch” kanalına “x” değişkeninin değerini yazar
- input prosesi: “ch” kanalından okuduğu değeri y adresine taşır
- Senkron iletişim: input/output proseslerinden önce yürütüleni diğeri yürütülene kadar askıya alınır. Eşleşme halinde, sonuç “y:=x” ataması şeklinde gerçekleşir (tamponlama yok).
- Ölümcül kilitlenme:



ATAMA PROSESİ

- Sentaks: *assignment* = *variable* := *expression*
- Hedef değişken ile ifadenin üreteceği sonuç değeri aynı tipten olmalıdır.
- İfade: değişken, sabit, operatör ve fonksiyon çağrıları içerebilir. Fonksiyon çağrıları sınırlıdır, yan etkileri bulunamaz ve yeni prosesler yaratamazlar.
- Atama eylemini yürüten proses bloke olamaz ve işlem tamamlanınca sona erer.
- Çoklu atama: birden fazla değişkene paralel atama yapılır

```
a, b, c := x, y + 1, z + 2
x, y := y, x
```

STOP ve SKIP PROSESLERİ

- Stop: hiç sonlanmayan NOP→ prosesin devamına engel olur (geri dönüşü olmayan bir askıya alınma).
- Hata durumlarında "stop" yürütülür.
- Örnek:

```
SEQ
  keyboard ? char
  STOP
  screen   ! char
```

Output prosesi hiç bir zaman yürütülmez.

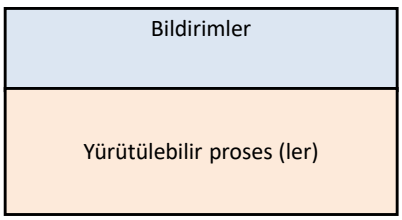
- Skip: hemen sona eren NOP→etkisiz proses

```
SEQ
  keyboard ? char
  SKIP
  screen   ! char
```

input prosesi yürütülür, skip hemen sonlanır ve ardından output prosesi çalışır.

BLOK PROSESLER

- Blok Proses: değişken ve kanal bildirimleri ve onları izleyen proses tanımları blok prosesi oluşturur.
- Değişken ve kanal bildirimleri kullanılacakları bağlamdan hemen önce tanımlanırlar.
- İç-içe blok tanımları mümkündür; bilinen tanımlı olma kuralları geçerli olur.



OCCAM BİLDİRİMLERİ

Veri Tipleri:

INT, BYTE, BOOL
INT16, INT32, INT64
REAL32, REAL64
} İlkel tipler

[100]INT
[32][32][8] BYTE
[]REAL64
} Diziler

Sabit Tanımları:

```
VAL INT max IS 50:
VAL INT double.max is 2*max:
VAL BYTE letter IS 'A':
VAL []BYTE string IS "Hello":
VAL [8]INT masks is [#01, #02, #04, #08, #10, #20, #40, #80]
```

OCCAM BİLDİRİMLERİ

Değişken bildirimleri:

```
INT16  a, b, time:
REAL64 X,Y:
BOOL flag:
[5]INT x:
[4][5]INT bigx:
```

Kanal bildirimleri:

```
CHAN OF INT q:
CHAN OF BYTE screen, keyboard:
[32] CHAN OF INT c,out: //iki kanal dizisi
PLACE keyboard AT 2:
```

Timer bildirimleri:

```
TIMER Big.Ben:
TIMER clockA, clockB:
[10] TIMER clocks:

Big.Ben?time
Big.Ben?AFTER t
```

"delayed input" : input işlemini Big.Ben timer değeri >t olana kadar askıya alır ve hemen sonra sonlandırır→prosesi belirli bir süre bekletmiş olur

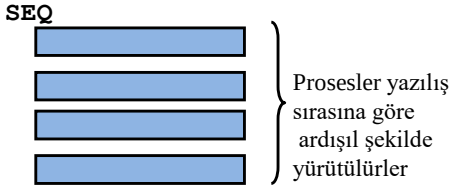
KURUCU PROSESLER (CONSTRUCTED PROCESSES)

- İlkel prosesleri değişik şekillerde biraraya getirip, geniş kapsamlı prosesler oluşturan yapılardır.
  - SEQ: ardışıl çalışmayı öngörür
  - WHILE: döngüsel çalışmayı öngörür
  - PAR: paralel çalışmayı öngörür
  - IF: koşullu çalışmayı öngörür
  - ALT: seçenekler arasından tercihli çalışmayı öngörür

SEQ PROSESİ

- SEQ (sequence): bu proses içerdiği proseslerin ardışıl sırada yürütülmesini öngörür.
- Prosesler "Q" karakterinin altına yazılır. Daha sola yazılmış ilk proses bu yapının dışında kalır.
- SEQ içindeki her proses sona ermelidir ki, bir sonraki başlayabilsin.
- SEQ kendisi de bir prosestir, içerdiği son proses de sona erince kendisi de sona erer.
- Sentaks:

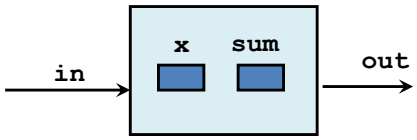
```
SEQ
  { process }
```



SEQ PROSESİ

Örnek:

```
SEQ
  in ? sum
  in ? x
  sum := sum + x
  out ! sum
```



Örnek:

```
SEQ
  SEQ
    screen ! '?'
    keyboard ? char
  SEQ
    screen ! char
    screen ! cr
    screen ! lf
```

→

```
SEQ
  screen ! '?'
  keyboard ? char
  screen ! char
  screen ! cr
  screen ! lf
```

TÜRETİCİ YAPI (replicator)

- Prosesleri belirli sayılarda türetip, çoğaltmak için kullanılır.
- SEQ, PAR, IF ve ALT için uygulanabilir.
- SEQ türeticisi:
- Sentaks:

sequence = SEQ *replicator*
process

replicator = *name* = *base* FOR *count*
base = *expression*
count = *expression*

- Örnek:

```
SEQ i = 0 FOR array.size
  stream ! data.array[i]
```

→

```
SEQ
  stream ! data.array[0]
  stream ! data.array[1]
```

base, base+1, …, base+count -1 kadar türetme

SEQ TÜRETİCİSİ

- Örnek:

```
SEQ i=0 FOR 10
  cnt!i
```

0,1,…9 değerlerini ard arda cnt kanalına gönderir.

- Özel durumlar:
  - Üst sınır "count" = 0 ise→SKIP yürütülür
  - Üst sınır "count" < 0 ise→STOP yürütülür
  - SEQ türeticisi yalnızca **bir** prosese uygulanabilir. Birçok prosese uygulayabilmek için ikinci bir SEQ'e gerek duyulur.
  - Örnek:

```
SEQ
  total:=0
  SEQ i=1 FOR N
    SEQ
      cnt?temp
      total:=total+temp
  SUM!total
```

WHILE PROSESİ

- WHILE, ilişkili olduğu lojik ifade TRUE olduğu sürece bir prosesin tekrarlanmasını sağlar.
- Sentaks:

loop = `WHILE` *boolean*
process

boolean = *expression*

`WHILE <boolean>`

"while prosesi"

```
WHILE A < 0  // pozitif bir a değeri
  in ? a     okuyana kadar tekrarla
```

Örnek:

```
CHAN OF INT in, out:
WHILE TRUE
  INT x:
  SEQ
    in ? x
    out ! 2*x
```

WHILE PROSESİ

- Örnek: 32(-32) büyük bir değer okuyana kadar, okunan değerin karesini hesapla ve gönder.

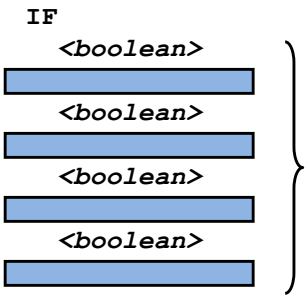
```
CHAN OF INT in, out:
INT A:
WHILE A<=1024
  SEQ
    in ? A
    A:=A*A
    out ! A
```

```
CHAN OF INT in, out:
INT A:
SEQ
  in ? A
  A:=A*A
  WHILE A<=1024
    SEQ
      out ! A
      in ? A
      A:=A*A
  out ! A
```

Ancak ilk anda A'nın değeri tanımsız!!

IF PROSESİ (conditional)

- IF prosesi, her biri bir lojik ifade ile korunan bir dizi prosesten oluşur. Lojik ifadeler yazım sırasına göre değerlendirilirler. Bir ifade True sonucunu veriyor ise, koruduğu proses yürütülür ve IF prosesi sona erer. Hiç bir ifade doğru değil ise, IF prosesi STOP yürütmüş gibi davranır.



IF PROSESİ

- Sentaks:

```
conditional     =  IF
                     { choice }
choice          =  guarded.choice
                |  conditional
guarded.choice  =  boolean
                     process
boolean         =  expression
```

- Örnekler:

```
IF
  x < y
    x := x + 1
  x >= y
    SKIP
```

```
IF
  x < y
    x := x + 1
```

```
IF
  x > y
    order := gt
  x < y
    order := lt
  TRUE
    order := eq
```

IF TÜRETİCİSİ

- IF prosesi de türetme yoluyla çoğaltılabilir.
- Örnek:

```
IF i = 1 FOR length
  string[i] <> object[i]
    found := FALSE
TRUE
  found := TRUE
```

```
IF
  string[1] <> object[1]
    found := FALSE
  string[2] <> object[2]
    found := FALSE
  TRUE
    found := TRUE
```

- String ve object dizilerinin elemanlarını karşılaştır.

PAR PROSESİ

- PAR prosesi içerdiği bir dizi prosesin paralel yürütülmesini öngörür. Tümü sonlanınca, PAR prosesi de sonlanır.

PAR

Sentaks:

```
PAR
  { process }
```

```
WHILE next <> eof
  SEQ
    x := next
    PAR
      in  ? next
      out ! x * x
```

PAR PROSESİ

- Kısıtlamalar:
  - Paralel prosesler kanallar üzerinden haberleşirler.
  - Değişkenler kendilerine değer atanmadıkları sürece paralel proseslerde yer alabilirler. Atama yoluyla veya input yoluyla değeri değişebilen değişkenlere izin verilmez.
  - Hatalı kullanım örneği: "mice" değişkeni

```
PAR
  SEQ
    mice := 42
    c ! 42
  c ? mice
```

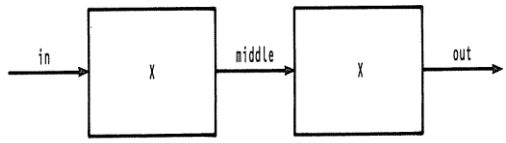
PAR PROSESİ

- Kısıtlamalar:
- Bir kanal birden fazla paralel proses içinde giriş/çıkış için kullanılamaz.
- Hatalı örnek: "c" kanalı kullanımı hatalıdır.

```
PAR
  c ! 0
  SEQ
    c ? x
    c ? y
  c ! 1
```

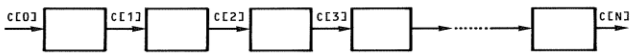
PAR PROSESİ:ÖRNEK

```
CHAN OF INT in, out, middle:
PAR
  INT X:
  WHILE TRUE
    SEQ
      in ? X
      middle ! X
  INT X:
  WHILE TRUE
    SEQ
      middle ? X
      out ! X
```



PAR TÜRETİCİSİ

- Türetme eylemi PAR prosesi için de uygulanabilir.
- Aynı özelliklere sahip olan bir dizi proses yaratılır.
- Örnek: 10 elemanlı bir tampon, her tampon elemanını taşıyan bir proses dizisi ile temsil edilir.
-




```
VAL INT N IS 10:
[N + 1]CHAN OF INT C:
PAR
  PAR P = 0 FOR N
    INT BufferElement:
    WHILE TRUE
      SEQ
        C[P] ? BufferElement
        C[P + 1] ! BufferElement
```

PAR TÜRETİCİSİ

- Türetme eylemi sadece bir PAR prosesine uygulanabilir.
- Bir grup proses türetilecek ise, bu durum belirtilmelidir.
- Örnek:

```
PAR  I=1 FOR 3          PAR
  PAR                     P(1)
    P(I)                  P(2)
    Q(I)        ⟶         P(3)
                          Q(1)
                          Q(2)
                          Q(3)
```

Tüm prosesler sona erdikten sonra türetilmiş PAR prosesi de sona erer.

Örnek: Kabarcık Sıralama

- Kabarcık sıralama algoritmasının paralel çözümü
- Sıralanacak N tamsayı değer için N adet paralel proses
- Her proses bir tamsayı değer tutar. Sol komşudan aldığı yeni değer sahip olduğundan daha küçük ise, kendininki ile yer değiştirip büyük olanı sağ komşusuna gönderir.
- Sıralanacak olan veriler ilk prosese gönderilir. "-1" veri sonunu belirtir. Veriler sonlandıktan sonra sıralı dizi son prosesten okunabilir.
- Bildirimler:

```
VAL INT FLUSH IS -1:
VAL INT N IS 30:
[N + 1]CHAN OF INT C:
```

Örnek: Kabarcık Sıralama

```
PAR
  -- other processes
  PAR I = 0 FOR N
    INT X,Y:
    SEQ
      C[I] ? X    -- read first value
      WHILE X <> FLUSH
        SEQ
          C[I] ? Y
          IF
            X > Y
              SEQ
                C[I + 1] ! X
                X := Y
            TRUE
              C[I + 1] ! Y
      C[I + 1] ! FLUSH   -- send on FLUSH value
```

PAR TÜRETİCİSİ

- Türetilmiş PAR ile yaratılan proseslerin sayısı türeticinin sonunda yer alan ifade tarafından belirlenir.
- Gerek duyulacak bellek alanının yürütme öncesinde hesaplanabilmesi için, derleme sırasında bir programda yer alan proses sayısının bilinmesi gerekir.
- Bu nedenle OCCAM yürütme sırasında dinamik olarak proses yaratılmasına izin vermez – yaratılacak olan proses sayısı derleme esnasında biliniyor olmalıdır.
- Türetilmiş PAR içindeki proses sayısını belirleyen ifade derleme sırasında hesaplanabilir olmalıdır; genellikle bir sabit değer kullanılır.

PAR TÜRETİCİSİ

- Bu durum bazı problemlerde kısıtlayıcı olabilir.

```
SEQ
  Start.Value ? N
  PAR P = 0 FOR N
    subprocess
```

Hatalı, çünkü N değeri derleme aşamasında bilinmemektedir.

- Ancak gerek duyulacak proses sayısının max değeri yürütme öncesinde bilinebilir ise (örnekteki N için bir tahminde bulunulabilir ise) küçük bir yürütme maliyeti ile bir çözüm üretilebilir.

PAR TÜRETİCİSİ

```
INT N:
VAL INT MAX IS 100:
SEQ
  Start.Value ? N
  IF
    N <= MAX
      SKIP
  PAR P = 0 FOR MAX
    IF
      P <= N
        subprocess
      TRUE
        SKIP
```

Eğer okunan N > MAX ise
IF hatalı olur→STOP (program durur)

MAX adet proses türet.
İlk N tanesi istenen prosesler, diğerleri (P>N :N+1, N+2,..,MAX) için SKIP prosesleri yaratılır, onlar da hemen sona ererler.

PROSES ÖNCELİĞİ: PRI PAR

- PAR ile oluşturulan proseslerin ne zaman canlanacakları önceden kestirilemez.
- Bazı durumlarda, bir prosesin diğerlerine oranla daha önce yürütülmesi gerekebilir. Bir prosese öncelik verilmesi, aynı işlemci üzerinde olmaları koşulu ile, çalışmaya hazır olan prosesler içinden yüksek öncelikli prosesin seçilmesini sağlar.
- Gerçek zaman problemlerinde, dış olaylara yanıt verecek proseslere yüksek öncelik tanımlanması gerekebilir.
- PRI PAR: proseslerin önceliklerini belirler; proseslerin metinde yer alma sırası aynı zamanda öncelik sırasını belirler.

PROSES ÖNCELİĞİ: PRI PAR

- Örnek:

```
PRI PAR
  P1          -- en yüksek öncelikli proses
  PAR
    P2
    P3        -- eşit ve orta öncelikli üç proses
    P4
  P5          -- en düşük öncelikli proses
```

- Her öncelik düzeyi için ayrı bir kuyruk gerektiği için PRI PAR içindeki öncelik düzeyi sayısı sınırlıdır. Genellikle iki düzey vardır.

PLACED PAR

- PLACED PAR: PAR ile tanımlanan proseslerin farklı işlemcilere atanacaklarını ve gerçek anlamda paralel yürütüleceklerini belirtir.
- Örnek:

```
PLACED PAR
  PROCESSOR 1
    P1
  PROCESSOR 2
    P2
```

- P1 prosesi 1 numaralı işlemci, P2 prosesi de 2 numaralı işlemciye atanıp yürütüleceklerdir.