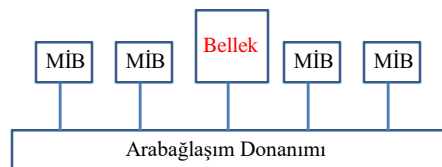


MESAJ AKTARIMI

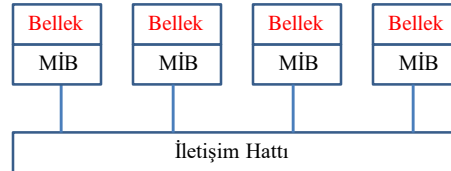
DAĞITILMIŞ MİMARİLER

- Dağıtılmış mimariler iki sınıfa ayrılır:
 - Sıkı bağlı (tightly coupled)
 - Gevşek bağlı (loosely coupled)
- Sıkı bağlı mimariler: İşlemciler bir ortak belleği paylaşır.



DAĞITILMIŞ MİMARİLER

- **Gevşek bağlı mimariler:** Her işlemcinin kendine ait bir yerel belleği vardır. Paylaşılan bir ortak bellek yoktur.
- İşlemciler birbirlerine bir iletişim hattı üzerinden bağlıdırlar.



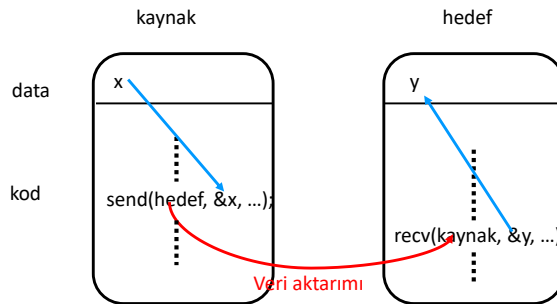
MESAJ AKTARIMI

- **Gevşek bağlı mimari:** İşlemciler iletişim hattı üzerinden aktarılan **mesajlar** aracılığı ile haberleşirler.
- Mesaj aktarımı paralel çalışan proseslerin iki temel problemine çözüm getirir:
 - **Bilgi paylaşımı:** mesaj alan proses mesaj içeriği ile belirli bir bilgiye sahip olur.
 - **Senkronizasyon:** mesajın alınabilmesi için önce gönderilmiş olması gerekir → işlemlerin gerçekleşme sırası üzerinde belirli bir sınırlama getirir, mesajlaşan prosesler zaman içinde senkronize olurlar.

MESAJLAŞMA İLKELLERİ

- **SEND/RECEIVE:**

- **send(hedef, data, ...)** “data”nın mesajın yürütüldüğü andaki değeri aktarılır. “hedef” mesajın varacağı yer üzerinde program denetimi sağlar.
- **receive(kaynak, &buf, ...)** “kaynak” kabul edilebilecek mesajlar üzerinde denetim sağlar. Mesaj içeriği “buf” adresine atanır.



MESAJLAŞMA İLKELLERİ

- Mesaj içeriği anlamlı herhangi bir şey olabilir:
 - Veri
 - Uzaktan prosedür çağırısı
 - Yürütülebilir kod...
- Mesajlar genellikle standart alanlardan oluşurlar:
 - Gönderen prosesin kimliği
 - Alıcı prosesin kimliği (yanıt dönecek ise)
 - Mesajın boyu
 - Veri tipi
 - Veri...

HABERLEŞME KANALLARI

- Hedef ve kaynak belirleyicilerine **haberleşme kanalı** adı verilir.
- Haberleşme kanalı mesaj aktarımının fiziksel ortamda nasıl yürütüleceği ile ilgilenmez.
- Haberleşme kanalı ilkelin lojik düzeyde gerçekleşmesine ilişkin ayrıntıları belirler.
- Haberleşme kanalları iki şekilde adlandırılabilir:
 1. Doğrudan adlandırma
 2. Dolaylı adlandırma

DOĞRUDAN ADLANDIRMA

- **Doğrudan Adlandırma** (Direct Naming): Mesaj aktarımına katılacak taraflar, kaynak ve hedef proseslerin isimlerini ilkel içinde kullanırlar.

Proses Q:

send (P, mesaj) : P prosesine bir mesaj gönder

Proses P:

receive (Q, mesaj) : Q prosesinden bir mesaj kabul et

- Adları verilen prosesler arasında bir lojik kanal oluşturulur.
- Taraflar karşılıklı olarak birbirlerinin isimlerini bilmek durumundadırlar. Proses isimleri biricik (unique) olmalıdır.

DOĞRUDAN ADLANDIRMA

- Örnek çalışma:

```
Process A
while (TRUE) {
  .. X verisi üret
  send ( B, X )
}
```

```
Process B
while (TRUE) {
  receive ( A, Z )
  ... veriyi kullan
}
```

- Yararları:

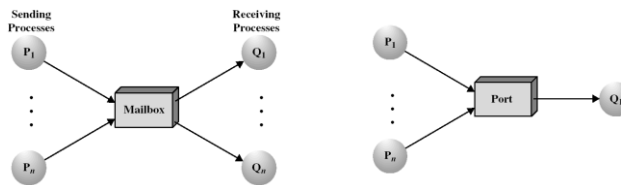
- Gerçeklenmesi kolay.
- Mesaj alışverişine yer alacak taraflar proses denetiminde olur.
- Pipeline (boru hattı) benzeri problemler için uygundur.

- Kısıtlamaları:

- Proses adları bilinmeli
- Esnek olmayan yapılara neden olur. İsim değişikliği sorun olur.

DOLAYLI ADLANDIRMA

- Dolaylı Adlandırma** (Indirect Naming): Mesajlar doğrudan proseslere değil de, prosesler arasında paylaşılan yapılara (**mailbox** veya **port**) gönderilir.



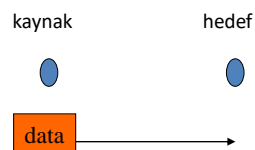
Mailbox/Port: Proseslerin içine mesaj yerleştirebildikleri ve başka proseslerin de mesajları çekebildikleri paylaşılan veri yapıları. Biricik bir isme sahip olmalılar. Sadece aynı yapıyı paylaşan prosesler haberleşebilirler.

MAILBOX / PORT

- Gerek duyarlarsa, bir proses çifti birden fazla mailbox aracılığı ile haberleşebilirler.
- Bir kanal tek yönlü veya çift yönlü aktarıma izin verebilir.
- Bir kanal birden fazla proses ile ilişkilendirilebilir.
 - Bire-birli haberleşme (one-to-one)
 - Bire-çoklu haberleşme (one-to-many)
 - Çoklu-tek haberleşme (many-to-one)
 - Çoklu-çoklu haberleşme (many-to-many)

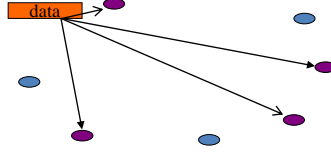
MAILBOX / PORT

- **Bire-bir haberleşme (point to point):**
 - Bir kanal sadece iki proses tarafından paylaşılır
 - Prosesler arasında özel bir kanal oluşturulmuş olur.



MAILBOX / PORT

- Bire-çoklu haberleşme (one to all):

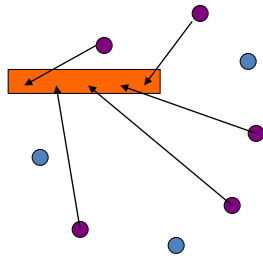


- Bir kanal çok sayıda proses tarafından paylaşılır.
- Bir proses kanala mesaj gönderir (send), diğer prosesler kanaldan mesaj alırlar (receive) → **Broadcast** benzeri haberleşme
- Receive yürütmüş ve bekleyen çok sayıda proses var ise:
 - Sadece bir prosesin receive isteğini kabul edip, işleme al.
 - Tüm receive isteklerini işleme al, veri hazır olunca içlerinden birini rastgele seç.

MAILBOX / PORT

- Çoklu-tek haberleşme (all to one):PORT

- Bir kanal çok sayıda proses tarafından paylaşılır.
- Bir çok proses kanala mesaj gönderirler (send), sadece bir proses kanaldan mesaj alır (receive).
- İstemci-Sunucu tipi uygulamalar için uygun bir yapı.



- Kanal bir sunucu proses ve çok sayıda istemci proses arasında paylaşılır.
- İstemciler hizmet isteklerini kanala (porta) gönderirler.
- Sunucu mesajın başlık alanından (header) istemcinin kimliğini belirleyebilir.

MAILBOX / PORT

- **Çoklu-çoklu haberleşme (all to all):MAILBOX**
 - Bir kanal çok sayıda proses tarafından paylaşılır.
 - Bir çok proses kanala mesaj gönderirler (send), bir çok diğer proses de kanaldan mesaj alır (receive).
 - Benzer hizmet veren çok sayıdaki sunucunun yer aldığı (server farm) istemci-sunucu tipi uygulamalar için uygun bir yapıdır.

MAILBOX / PORT

- **MAILBOX SAHİPLİĞİ:** erişim haklarını belirler.
- Mailbox bir **prosese** veya **sisteme** ait olabilir.
- **Proses aittir:**
 - Mailbox bir proses aittir ise, sadece o proses mesaj alabilir (receive).
 - Aynı mailbox'ı paylaşan diğer prosesler sadece mesaj gönderebilirler.
 - Proses yaratılırken mailbox yaratılır; proses sonlanınca da yok olur.
 - Sonlanmış bir proses aittir mailbox'a mesaj gönderen proses bir sinyal ile uyarılır.
 - Veya, proses uygun gördüğü anda **create_mailbox/destroy_mailbox** çağrılarını ile yapıyı yaratır veya sonlandırır.

MAILBOX / PORT

- **Sisteme ait:**
 - Mailbox bir proses ile ilişkilendirilmemiştir, kendi başına var olur.
 - Paylaşan prosesler sonlansa bile varlığını sürdürür.
 - Bir proses, haberleşme ilkelerini yürütmeden önce dinamik olarak mailbox'a bağlanmak zorundadır.

GERÇEKLEME AYRINTILARI

- Tamponlama yeteneği
- Doğrudan / Dolaylı haberleşme
- Senkron/Asenkron haberleşme

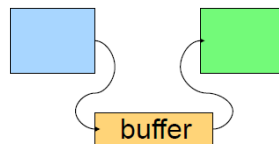
TAMPONLAMA

- **TAMPONLAMA KAPASİTESİ:** kanalda (mesaj kuyruğunda) geçici olarak saklanabilen mesaj sayısı
- **Sıfır kapasite:**
 - Mesaj kuyruğu boyu sıfırdır.
 - Gönderici, alıcı receive yürütene kadar bekler; mesaj doğrudan gönderenin bellek alanından alıcının bellek alanına taşınır.



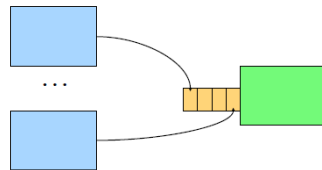
TAMPONLAMA

- **Sınırlı kapasite:**
 - Mesaj kuyruğunun belirli bir kapasitesi vardır.
 - Kuyruk dolu olmadığı sürece mesajlar kabul edilir. Kuyruk dolu ise gönderen bekler.
- **Sınırsız kapasite:**
 - Sonsuz sayıda mesaj kabul edilir (sanal bellek kullanılır).
 - Mesaj gönderen hiç bir zaman beklemez.



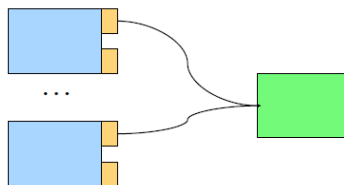
DOĞRUDAN HABERLEŞME

- **Sadece alıcıda bir tampon alanı**
 - Birden fazla gönderici alıcıya mesaj gönderebilir
 - Belirli bir göndericiden gelen mesajı seçmek için tüm tamponun taranması gerekir



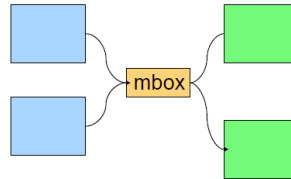
DOĞRUDAN HABERLEŞME

- **Her göndericide bir tampon alanı**
 - Bir gönderici birden fazla alıcıya mesaj gönderebilir



DOLAYLI HABERLEŞME

- **Posta Kutusu** yapısı kullanılır
 - Çoklu (many-to-many) haberleşme olanağı sağlar
 - posta kutusunu yarat/aç/kapat ... gerekir
 - Tampon: dışlamalı erişim gerekir
 - Mesaj boyu: kısıtlı olabilir, birden fazla paket kullan



SENKRON / ASENKRON İLETİŞİM

- İletişim, ilkelerin neden olduğu **gecikmeler** göz önüne alınarak iki sınıfta incelenir: **senkron** ve **asenkron**.
- **Gecikme**: send/receive çağrılarının gerçekleşme aşamalarının bu çağrıları yürüten proseslerin kontrol akışları üzerinde yarattığı etki.
- Çağrı iki aşamada gerçekleşir:
 1. Send/receive çağrısının iletişim altyapısına iletilmesi
 2. Çağrıya ilişkin adımların yürütülmesi (mesajın gönderilmesi veya alınması)
- **Gecikmesiz (nonblocking)**: Çağrı 1. aşamada sonlanır
- **Gecikmeli (blocking)**: Çağrı 1. ve 2. adımların tamamlanmasını bekler.

SENKRON İLETİŞİM (blocking send/blocking receive)

- Send ve Receive ilkellerinin her ikisi de gecikmelidirler.
 - **Gönderici proses**, alıcı proses mesajı alana kadar askıya alınır.
 - Alıcı bir mesaj al çağrısı yürütmemiş ise bekle
 - Veri aktarımını başlat
 - Veri gönderici belleğinden aktarılanaya kadar bekle
 - Alıcı receive adımını tamamlayıp alındı mesajı gönderene kadar bekle

send(dest, type, msg) ↘

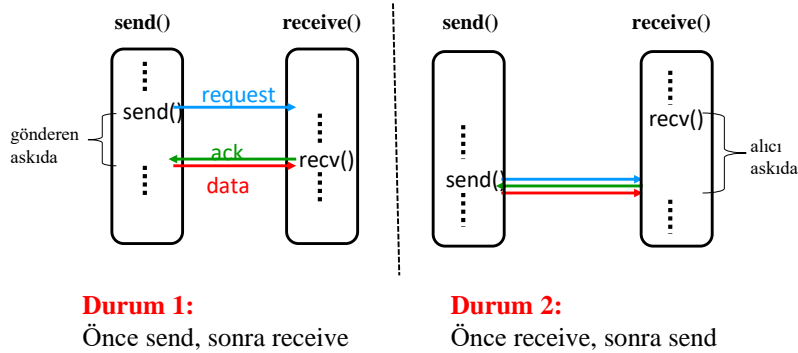
- **Alıcı proses**, bir mesaj gönderilene kadar askıya alınır.
 - Bir mesaj hazır olana kadar bekle ve mesajı geri getir

recv(src, type, msg)

SENKRON İLETİŞİM (blocking send/blocking receive)

- Prosesler **zaman içinde senkronize olurlar** (rendez-vous).
- Gönderici **mesajının alındığına** emin olur. Alıcı mesaj **içeriğinin güncel** olduğuna emin olur.
- Gerçeklemesi kolay.
- Kısıtlayıcı:
 - Gönderici mesajı kabul edilene kadar beklemek yerine çalışabilir.
 - Alıcı bir mesaj hazır olana kadar beklemek yerine çalışabilir.

SENKRON İLETİŞİM



ÖLÜMCÜL KİLİTLENME

- P1 ve P2 prosesleri verilerini değiş tokuş edeceklerdir:

P₁:

```
send(P2, data1);
receive(P2, data2);
```

P₂:

```
send(P1, data2);
receive(P1, data1);
```

Veya:

P₁:

```
receive(P2, data2);
send(P2, data1);
```

P₂:

```
receive(P1, data1);
send(P1, data2);
```

- Prosesler aynı işi yaptıkları halde aynı kodu yürütemezler.

ÖLÜMCÜL KİLİTLENME

- P1 ve P2 prosesleri send ve receive çağrılarını farklı sıralarda yürütmek zorundadırlar:

P₁:

```
send(P2, data1);
receive(P2, data2);
```

P₂:

```
receive(P1, data1);
send(P1, data2);
```

ASENKRON İLETİŞİM (non_blocking send/blocking receive)

- Send ilkeli gecikmesiz ancak Receive gecikmelidir.
 - **Gönderici proses**
 - İletişim altyapısı tarafından çağrı kabul edilene kadar bekle
 - Veri aktarımını başlat ve veri sistemin mesaj kuyruğuna kopyalandıktan sonra geri dön. Alıcının mesajı kabulü için beklenmez.
 - Çağrının tamamlanması: Çağrının geri getireceği bir dönüş kodu üzerinden mesajın akıbetinin sorgulanması gerekebilir eğer uygulama bu bilgiye gerek duyuyor ise
 - Örnek uygulama

```
status = async_send( dest, type, msg )
...
if !send_complete( status )
    wait for completion;
...
use msg data structure;
...
```

ASENKRON İLETİŞİM (non_blocking send/blocking receive)

- Send ilkeli gecikmesiz ancak Receive gecikmelidir.
 - **Alıcı proses**
 - Bir mesaj hazır olana kadar askıda bekle ve mesajı geri getir.
 - Örnek uygulama: “sunucu receive ile mesaj için bekler, gelen mesajı hizmeti sunacak olan bir işçi ipliğe aktarır”, “..” işlemleri tekrarlar.
 - Eğer mesaj gelene kadar bloke olmak istenmez ise, iletişim altyapısının sağlayacağı bir dönüş bilgisi üzerinde sorgulama yaparak receive ilkelinin yürütülmesi ertelenebilir; bu arada başka işlemler yürütülerek bekleme ile harcanacak zaman değerlendirilebilir.


```
while ( probe(src) != HaveMSG )
  wait for msg arrival
recv( src, type, msg );
consume msg;
```

ASENKRON İLETİŞİM (non-blocking send/blocking receive)

- **Yararı:** Hesaplama ve iletişim işlemleri zaman içinde çakıştırılabilir → **maximum paralellik** sağlanır.
- **Kısıtlayıcı:**
 - Prosesler **senkronize olmazlar**.
 - Alınan **mesajın güncelliğinden** emin olunamaz.
 - Mesajın iletişim sistemine kopyalanması gerekir, zaman alır.
 - Mesajın alıcıya ulaşıp ulaşmadığı ayrıca kontrol edilmesi gerekir (eğer bu bilgiye ihtiyaç varsa).

PROSES SENKRONİZASYONU:SEMAFOR

- Bir **mailbox semafor** olarak kullanılabilir:

“non-blocking send”+“blocking receive” 
 P(sem) → receive
 V(sem) → send

- Başlangıç durumu:

create_mailbox (mutex) // mailbox yarat
 send (mutex, boş-mesaj) // boş mesaj gönder

- Proses P_i:

```
while (TRUE) {
    receive (mutex, boş-mesaj); //mesajı okuyan proses KB'e
    ..... //kritik bölüm      // ilerleme hakkı kazanır
    send (mutex, boş-mesaj);
}
```

ÜRETİCİ TÜKETİCİ PROBLEMİ

- Tek üretici ve tek tüketici durumu

```
void producer(void)
{
    while (TRUE)
    {
        ...
        produce item;
        ...
        send( consumer, item );
    }
}
```

```
void consumer(void)
{
    while (TRUE)
    {
        recv( producer, item );
        ...
        consume item;
        ...
    }
}
```

ÜRETİCİ TÜKETİCİ PROBLEMİ

- **İkili semafor:** bir tek mesaj ile gerçekleştirilebilir (bir önceki örnek)
- Üretici Tüketici probleminde tampon alanın kapasitesi (n) bir sayma semaforuna atanan ilk değer olacaktır. Bu durumda, n elemanlı bir tampon oluşturmak için n adet boş mesaja gerek duyulur.
- **Sayma semaforu:** semaforun ilk değeri n: tampon alanın boyutu → n adet mesaj ile gerçekleştirilebilir.
- Çözümde **iki mailbox** kullanılır:
 - mailbox “üretme_izni”
 - mailbox “tüketme_izni”

ÜRETİCİ TÜKETİCİ PROBLEMİ

- İlkendirme işlemleri:

```
mesaj = "boş mesaj"
n = tampon eleman sayısı
create_mailbox ( üretme_izni );
create_mailbox ( tüketme_izni );
for (i=0; i<n; i++) send (üretme_izni, mesaj);
```

- “üretme_izni” ← n adet boş mesaj gönderilir (boş tampon)
- Üretici ve Tüketici prosesleri bu adımdan sonra çalışmaya başlayabilirler.

ÜRETİCİ TÜKETİCİ PROBLEMİ

- Proses Üretici:

```
while (TRUE) {  
    receive (üretme_izni, mesaj); //boş mesaj al  
    mesaj = "yeni veri "  
    send (tüketme_izni, mesaj); // dolu mesaj gönder  
}
```

- Proses Tüketici:

```
while (TRUE) {  
    receive (tüketme_izni, mesaj); // dolu mesaj al  
    "mesaj içeriğini kullan "  
    send (üretme_izni, mesaj); //boş mesaj gönder  
}
```