YZM 3102
İşletim Sistemleri

Yrd. Doç. Dr. Deniz KILINÇ
Celal Bayar Üniversitesi
Hasan Ferdi Turgutlu Teknoloji Fakültesi
Yazılım Mühendisliği

BÖLÜM - 4

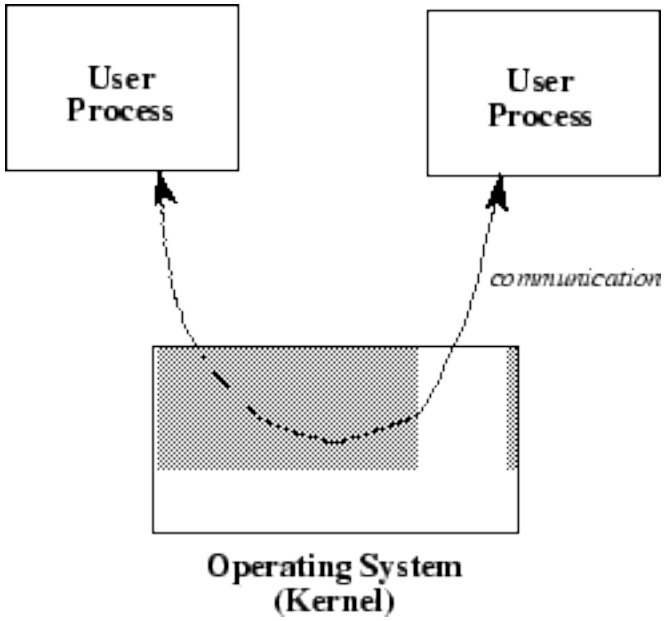
Bu bölümde,

- IPC (Interprocess Communication)
- Shared Memory
- Message Passing
- IPC Örnekleri
- IPC POSIX Shared Memory
- C# Windows Soket Programlama
- Pipe

konularına değinilecektir.

IPC (Prosesler Arası İletişim)

- İşletim sisteminde aynı anda çalışan (concurrent) prosesler, birbirinden **bağımsız (independent)** veya **işbirliği** (**cooperating**) içerisinde olabilirler.
- Birbirlerinden etkilenmeyen ve birbirleri ile **veri paylaşmayan** prosesler, bağımsız proseslerdir.
- *Tam aksine* birbirlerinin sonucundan etkilenen veya birbirleri ile veri paylaşan prosesler işbirliği içerisindedirler.



Interprocess Communication

IPC (Prosesler Arası İletişim) (devam…)

- Proseslerin işbirliği içerisinde olmalarının **temel 4 nedeni** (avantajları) vardır:
  1. **Bilgi Paylaşımı:** Sistemde birden çok kullanıcı aynı bilgiye (dosya, bellekteki bir bilgi) erişmeye çalıştıkları için bu bilgiye **concurrent** erişim için bir ortam oluşturulmalıdır.
  2. **Hesaplama Hızında Artış:** Birden fazla işlemcili veya çok çekirdekli bir bilgisayarda, bir işin daha **hızlı** sonlanmasını istiyorsak, işi *daha küçük alt işlere* bölerek **paralel** çalışmalarını sağlayabiliriz.

IPC (Prosesler Arası İletişim) (devam…)

3. **Modülerlik:** Sistemi modüler şekilde tasarlamak isteyebiliriz. Örneğin, sistem fonksiyonlarını prosesler ve iş parçacıkları şeklinde tasarlama.
4. **Uygunluk:** Her kullanıcı birden fazla görev üzerinde çalışıyor olabilir. Örneğin bir kullanıcı aynı anda müzik dinleyip, bir şeyler yazıp, mesaj alıyor olabilir.

- Proseslerin işbirliği içerisinde olabilmeleri için proseslerin bilgi paylaşımına **imkan sağlayacak** bir **IPC (Interprocess Communication)** mekanizmasına ihtiyaç vardır.

IPC (Prosesler Arası İletişim) (devam…)

IPC dezavantajları:

- Veri uyumsuzlukları
- Deadlock problemleri
- Artan karmaşıklık
- Proses senkronizasyon sorunları

IPC (Prosesler Arası İletişim) (devam…)

- IPC, **prosesler veya thread'lerin** OS tarafından sağlanan mekanizmalarla veri değiş-tokuşu, veri paylaşmalarıdır.
- Prosesler aynı makinede olabilecekleri gibi, ağ yoluyla bağlı dağıtık makineler de olabilir.
- Pipes, named pipes, message queueing, semaphores, shared memory ve sockets IPC yöntemleridir.

IPC (Prosesler Arası İletişim) (devam…)

- IPC'de baz alınan 2 temel model bulunmaktadır:
  1. Paylaşımlı Bellek (Shared Memory- SM): *Bellekteki bir bölge* **paylaşımlı** olarak belirlenir. Prosesler bu bölgeye bilgi yazar ve bilgi okurlar.
  2. Mesaj Geçişi (Message Passing - MP): İşbirliği içerisinde olacak *proseslerin birbirleriyle mesaj değiş tokuşu yapmaları* sonucunda **iletişim gerçekleşir**.
- Her iki model de işletim sistemlerinde **sıkça kullanılmaktadır**.

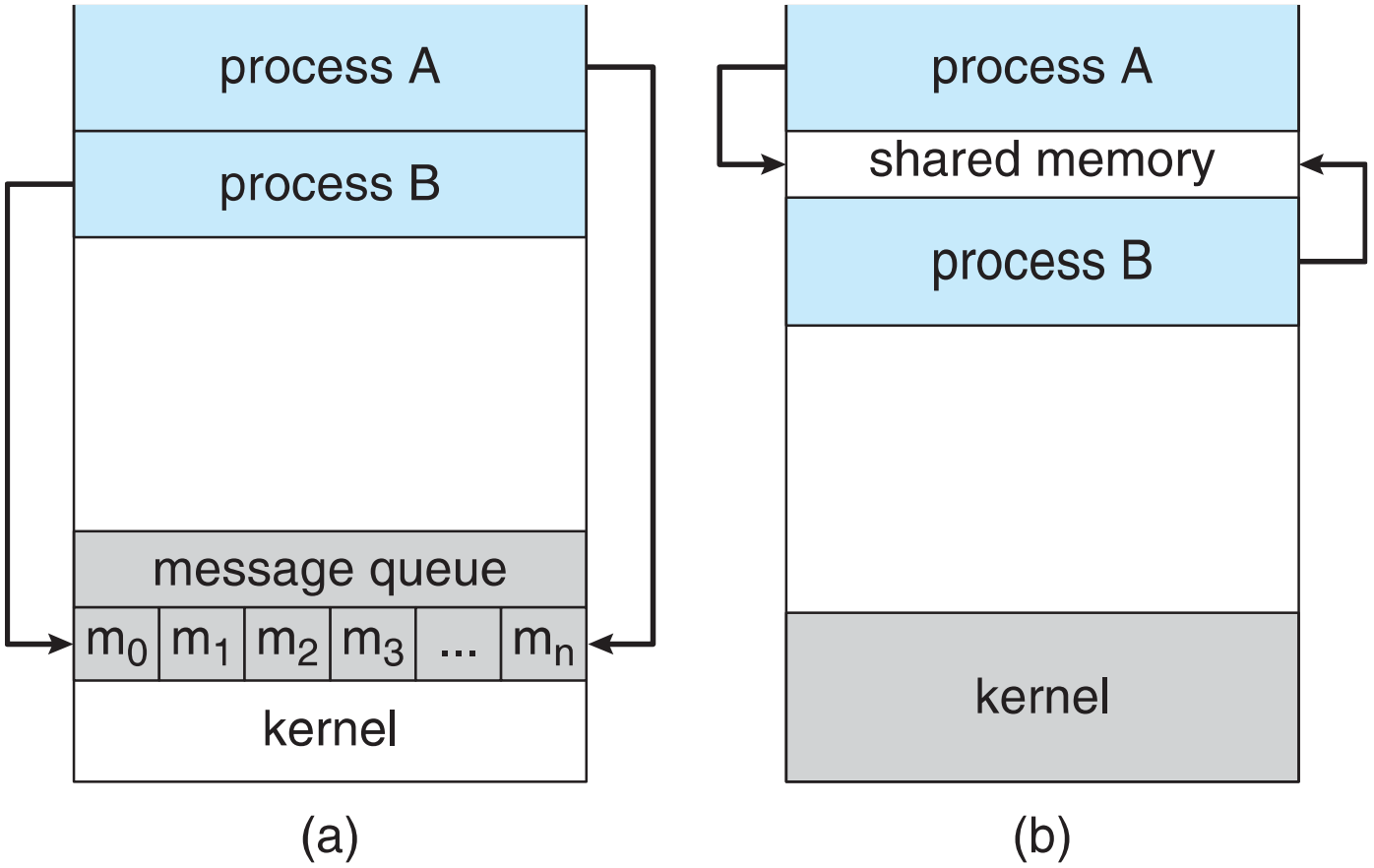
IPC (Message Passing) (devam…)

- **Mesaj geçiş yöntemi (MP)**
  - MP, bir IPC türüdür. Producer proses (producer/consumer bağlamında) tipik olarak mesaj göndermek için **send()** sistem çağrısını ve consumer process mesajları almak için **receive()** sistem çağrısını kullanır. Bu sistem çağrıları senkron/asenkron olabildikleri gibi, aynı makinede veya NW üzerinden *farklı makinelerde* koordinasyonu sağlamak için kullanılabilirler.
  - Daha az miktarda verinin paylaşılacağı özellikle, Ağ gibi *Dağıtık sistemlerde*, (örneğin chat, client/server mimarili database'ler vs) kullanılan bir yöntemdir.
  - Implementasyonu genel olarak daha basittir.
  - Bir proses ötekine mesaj göndereceği/alacağı zaman mesajı kernel'a iletir (send system call, receive system call) kernel bu mesajı diğer prosese iletir. Kernel bu süreçte aktiftir.

IPC (Message Passing) (devam…)

- **Paylaşımlı bellek yöntemi (SM)**
  - SM prosesler arasında iletişim sağlamak adına bir hafızanın ortak kullanılmasıdır. Bir proses (örneğin RAM'de) bir alan oluşturduğunda OS'un SM desteği ile (normalde prosesleri izole eder!) diğer prosesler **o alana erişip**, veri paylaşabilirler. Bu işlem MP yöntemine göre **daha hızlı olsa** da, verinin aynı anda erişilip güncellenmesi, *proseslerin aynı makine de olma zorunluluğu* yöntemin **zayıf taraflarıdır**.
  - Mesaj geçiş yöntemi sistem çağrıları ile kernel üzerinden gerçekleştirildiği için, paylaşımlı bellek mesaj geçiş yöntemine göre *daha hızlı çalışır*.
  - Paylaşımlı bellek ise **sadece paylaşımlı belleği oluştururken system call (kernel tarafı)** kullanır.
  - **Not :** Thread'ler default olarak «shared memory» kullanırlar.

IPC (Prosesler Arası İletişim) (devam…)



Shared Memory- SM

- SM bölgesi, bu bölge segmentini **yaratan** prosesin adres alanında (memory space) yer alır.
- Bu paylaşımlı bölgeyi kullanarak *iletişim kurmak isteyen prosesler*, bu adres alanına **bağlanmak zorundadırlar.**
- Normal şartlarda işletim sistemi bir prosesin diğer prosesin **bellek alanına erişmesini** önlemeye çalışır.
- Ancak SM ile haberleşme yapılmak isteniyorsa, bu tarz bir erişimin 2 veya daha fazla proses için gerçekleştirilebilir olması gerekmektedir.

Shared Memory- SM (devam…)

- Verinin içeriği ve boyutu prosesler tarafından belirlenir. OS'un herhangi bir kontrolü yoktur.
- Ayrıca, *aynı anda aynı bölgeye yazma işleminin gerçekleştirilemiyor olması* da proseslerin sorumluluğundadır.
- Paylaşımlı bellek yöntemini kullanarak *işbirliği halindeki prosesleri* **kavramsal olarak anlatmak** için

Producer - Consumer (Üretici - Tüketici)
problemine / metaforuna sıkça başvurulur.

Producer-Consumer Problemi

- **Producer (Üretici):** Sürekli bilgi üretir.
- Consumer (Tüketici): Producer tarafından üretilen bilgiyi sürekli tüketir.
- **Örneğin:** Client-server mimarisinde server producer, client ise consumer olarak düşünülebilir (Web server-web client).
- Bu probleme çözüm bulmak için **Shared Memory** yöntemi kullanılabilir.
- Producer ve consumer proseslerinin concurrent çalışabilmesi için bellekte geçici itemların bulunduğu erişilebilir bir bellek alanı olması (**buffer**) gerekmektedir.

Producer-Consumer Problemi (devam…)

- **Producer** buffer'a bir item eklerken, **consumer** buffer'dan bir item silebilmelidir.
- Bu noktada ***proseslerin senkronizasyonu*** önemlidir. Henüz **üretilmeyen** bir item, **tüketilmeye** çalışılmamalıdır.
- İki tür buffer bulunmaktadır:
  - **Unbounded (Sınırsız) buffer:** Size limiti yoktur. Buffer dolmaz.
  - **Bounded (Sınırlı) buffer:** Size limiti vardır. Buffer dolduğunda producer beklemelidir.

Prod.-Cons. Problemi SM Çözümü

- Paylaşımlı bounded buffer **dairesel bir dizidir**.
- *İki mantıksal değişken:*
- **in:** Next free position
- **out:** First full position
- If **(in == out) buffer_boş**
- If (((in + 1) % BUFFER_SIZE) == out) **buffer_dolu**

```
#define BUFFER_SIZE 10
typedef struct {
    . . .
} item;

item buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
```

Prod.-Cons. Problemi SM Çözümü (devam…)

Producer Prosesi

```
item next_produced;
while (true) {
   /* produce an item in next_produced */
   while (((in + 1) % BUFFER SIZE) == out)
       ; /* do nothing, buffer is full*/
   buffer[in] = next_produced;
   in = (in + 1) % BUFFER SIZE;
}
```

Prod.-Cons. Problemi SM Çözümü (devam…)

Consumer Prosesi

```
item next_consumed;
while (true) {
        while (in == out)
        ; /* do nothing, buffer is empty*/
        next-consumed = buffer[out];
        out = (out + 1) % BUFFER SIZE;

        /* consume the item in next consumed */
}
```

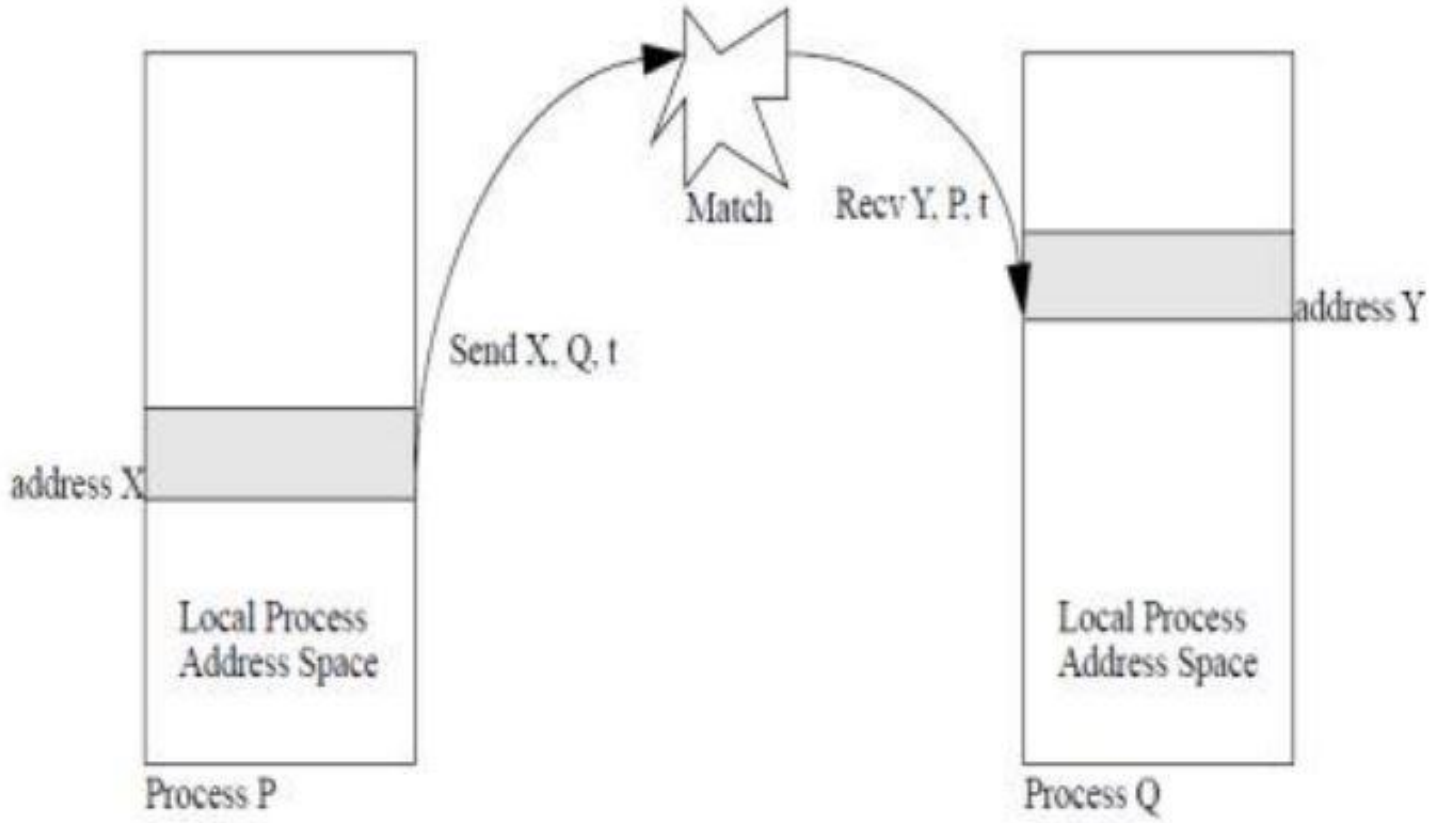
Prod.-Cons. Problemi SM Çözümü

- Değiştirilebilir / Geliştirilebilir bir çözüm.
- Üzerinde çalışılması lazım.
- Producer ve consumer proseslerinin aynı anda **senkronize** bir şekilde nasıl çalışacakları (paylaşımlı belleğe aynı anda ulaşarak yazma ve okuma işlemi) **anlatılmamıştır**.
- **Proses Senkronizasyonu** bölümünde detaylı olarak anlatılacaktır.

Message Passing – MP

- MP mekanizması, proseslerin aynı adres alanını paylaşmalarını **gerektirmeden** haberleşmelerini ve bilgi paylaşmalarını **sağlar**.
- Dağınık mimaride, *bir ağ altyapısında birbirine bağlı bulunan* farklı bilgisayarlardaki proseslerin haberleşmesinde kullanılır.
- **Örneğin:** İnternet chat programı
- MP iki temel operasyona sahiptir:
  - `send(message)`
  - `receive(message)`
- Message boyutu, *sabit* ve *değişken* olabilir.

Message Passing – MP (devam…)



Message Passing – MP (devam…)

- Eğer P ve Q prosesleri *haberleşmek istiyorlarsa*
  - P ve Q birbirlerine **mesaj gönderip** birbirlerinden **mesaj alabilmelidir**.
  - Bunun için P ve Q arasında **haberleşme (communication) linki** kurulmalıdır.
- Haberleşme linki **fiziksel** veya **mantıksal** olabilir.
- Fiziksel link:
  - shared memory,
  - HW bus veya
  - Network katmanında olabilir.

Message Passing – MP (devam…)

- **send()** ve **receive()** operasyonlarını içeren mantıksal bir link aşağıdaki implementasyon türlerini içerebilir (Aynı zamanda çözülmesi gereken 3 temel problem olarak da görülebilir):
  - *Direk* veya *İndirek (Dolaylı)* haberleşme - **Naming**
  - *Senkron* veya *Asenkron* haberleşme
  - *Otomatik* veya *Açık/Explicit Tamponlama* (buffering)

MP Implementation Questions

- How are links established?
- Can a link be associated with more than two processes?
- How many links can there be between every pair of communicating processes?
- What is the capacity of a link?
- Is the size of a message that the link can accommodate fixed or variable?
- Is a link unidirectional or bi-directional?

MP – Direk Haberleşme (devam…)

- Haberleşmek isteyen her proses **gönderici ve alıcı ismini** *belirtmek* zorundadır.
- Bu tasarımda send() ve receive() operasyonları aşağıdaki gibidir:
  - **send(P, message):** P'ye mesaj gönder.
  - **receive(Q, message):** Q'dan mesaj al.
- Haberleşme linkinin **özellikleri** aşağıdaki gibidir:
  - Haberleşmek isteyen her proses çifti arasında **otomatik olarak sadece (ve kesin) bir link** kurulur.
  - Bir **link** *sadece iki prosesle* ilişkilendirilir.

MP – Direk Haberleşme (devam…)

- Gönderici ve **alıcı** proseslerinin birbirlerinin **isimlerini bilmesi gerektiği** bu tasarım, **simetrik** tasarımdır.
- Bu tasarımın diğer bir varyasyonu da **asimetrik** olup send() ve receive() operasyonları aşağıdaki gibidir:
  - **send(P, message):** P'ye mesaj gönder.
  - **receive(id, message):** Herhangi bir prosesten mesaj al. Haberleşme kurulduğunda **id** değişkenine haberleşmenin gerçekleştiği, mesajın alındığı prosesin ismi atanır.

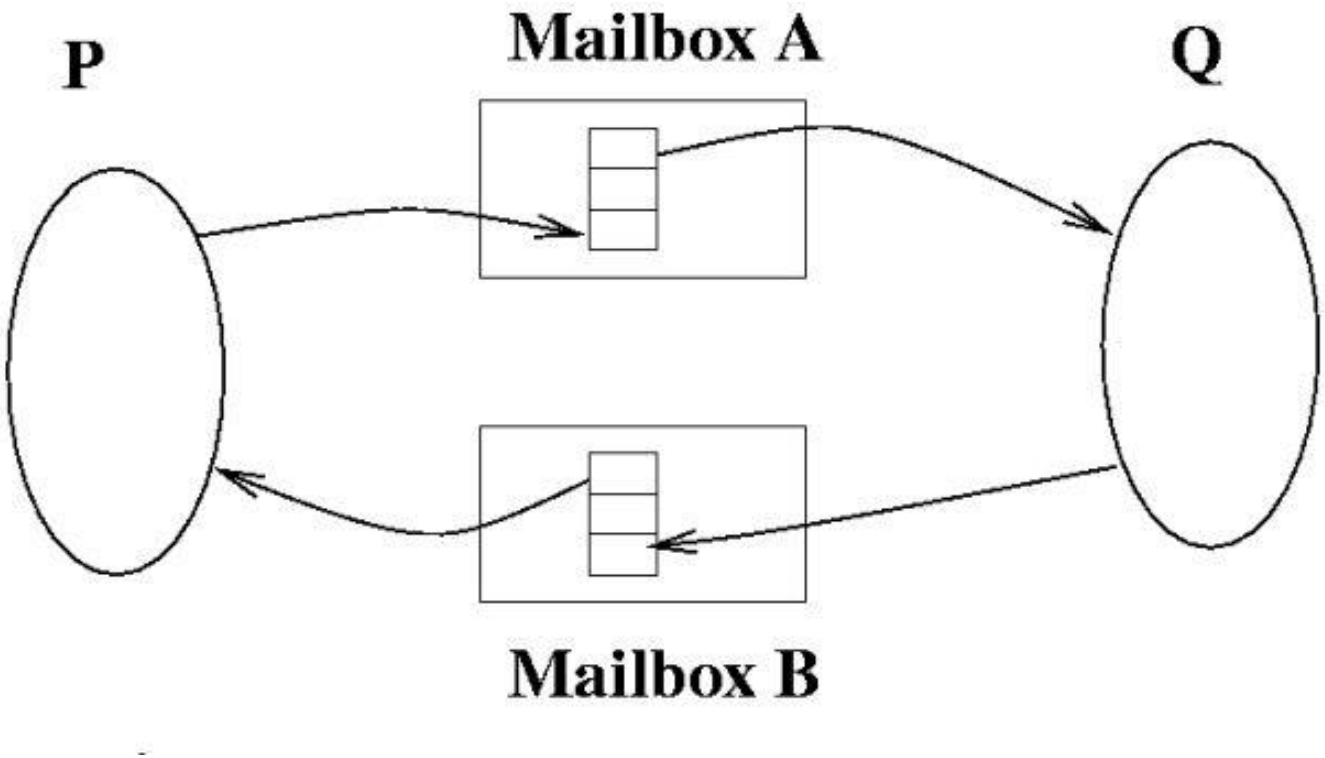
MP – Dolaylı Haberleşme

- Mesajlar port veya posta kutularından alınır veya buralara gönderilir.
- Her posta kutusu tek bir tanımlayıcıya sahiptir.
- Prosesler, paylaşılmış bir posta kutusuna sahipse *iletişim kurabilirler*.
- **Haberleşme linkinin özellikleri şunlardır :**
  - Bir link, ikiden fazla proses ile ilişkilendirilebilir.
  - Her bir proses çifti *birden fazla bağlantıya sahip olabilir*.
  - Bağlantı tek yönlü ya da çift yönlü olabilir.

MP – Dolaylı Haberleşme (devam…)

- Dolaylı haberleşmedeki operasyonlar aşağıdaki gibidir:
  - Yeni bir posta kutusu **oluştur**
  - Posta kutusu aracılığıyla mesaj gönder ve al
  - Posta kutusunu **yok et**
  - İletişim basitçe şu şekilde gerçekleşir:
    - **send(A, message):** A'nın posta kutusuna bir mesaj gönder
    - **receive(A, message):** A'nın posta kutusundan bir mesaj al.

MP – Dolaylı Haberleşme (devam…)



MP – Dolaylı Haberleşme (devam…)

- **Posta kutusu paylaşımı**
  - P_1, P_2, ve P_3 A posta kutusunu paylaşır.
  - P_1, gönderir; P_2 *ve* P_3 alır.
  - Kim mesajı alır?
- **Çözüm**
  - En çok iki proses ile ilişkili linke izin verilir.
  - Mesaj alma işlemi için sadece bir prosese izin verilir.
  - Sistem alıcıyı kendi isteğine bağlı olarak seçer.
  - Gönderici, alıcının kim bildirir.

MP – Dolaylı Haberleşme (devam…)

- **Tamponlama**
- Mesajlar kuyruklar üzerinden iletilirler. 3 tip kuyruk kapasitesinden bahsetmek mümkündür :
  - Sıfır kapasite – Mesajlar kuyrukta saklanmazlar. Dolayısıyla mesaj göndericiler, mesaj alıcılar mesaj kabul edene kadar beklemek zorundadırlar.
  - Sınırlı kapasite – Önceden tanımlı sınırlı bir kuyruk kapasitesi mevcuttur. Gönderici eğer kuyruk dolu ise beklemelidir.
  - Sınırsız kapasite – Kuyruk teoride sınırsız kapasiteye sahiptir, göndericiler hiçbir zaman beklemek durumunda kalmazlar.

IPC Örneği: POSIX Shared Memory

- Proses ilk olarak shared memory segmentini yaratır.
- ***shm_open()*** sistem çağrısı kullanılır, parametreleri aşağıdaki gibidir:
  - name: Shared memory ismi
  - O_CREAT: Shared memory objesi yoksa yarat
  - O_RDRW: Shared memory objesini okumak ve objeye yazmak için kullan
  - 0666: Dizin hakkını belirtir. Yazabilir ve okuyabilir.

```
shm_fd = shm_open(name, O_CREAT | O_RDRW, 0666);
```

IPC Örneği: POSIX Shared Memory (devam…)

- Shared memory objesi yaratıldıktan sonra ***ftruncate()*** fonksiyonu kullanılarak objenin size'ı set edilir.
  - Size: 4096 byte olarak set ediliyor.

```
ftruncate(shm_fd, 4096);
```

- Daha sonra ***mmap()*** fonksiyonu kullanılarak, shared memory objesine sahip memory-mapped dosyası yaratılır.
- Shared memory objesine yazmak için aşağıdaki fonksiyon kullanılır.

```
sprintf(sh_mem, "Message");
```

IPC POSIX SHARED MEMORY **PRODUCER**

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <sys/shm.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/mman.h>
int main() {
        /* the size (in bytes) of shared memory object */
        const int SIZE = 4096;
        /* name of the shared memory object */
        const char *name = "OS";
        /* strings written to shared memory */
        const char *message_0 = "Hello";
        const char *message_1 = "World!";
        /* shared memory file descriptor */
        int shm_fd;
        /* pointer to shared memory obect */
        void *ptr;
        /* create the shared memory object */
        shm_fd = shm_open(name, O_CREAT | O_RDWR, 0666);
        /* configure the size of the shared memory object */
        ftruncate(shm_fd, SIZE);
        /* memory map the shared memory object */
        ptr = mmap(0, SIZE, PROT_WRITE, MAP_SHARED, shm_fd, 0);
        /* write to the shared memory object */
        sprintf(ptr,"%s",message_0);
        ptr += strlen(message_0);
        sprintf(ptr,"%s",message_1);
        ptr += strlen(message_1);
        return 0;
}
```

IPC POSIX SHARED MEMORY **CONSUMER**

```c
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/shm.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/mman.h>
int main() {
        /* the size (in bytes) of shared memory object */
        const int SIZE = 4096;
        /* name of the shared memory object */
        const char *name = "OS";
        /* shared memory file descriptor */
        int shm_fd;
        /* pointer to shared memory obect */
        void *ptr;
        /* open the shared memory object */
        shm_fd = shm_open(name, O_RDONLY, 0666);
        /* memory map the shared memory object */
        ptr = mmap(0, SIZE, PROT_READ, MAP_SHARED, shm_fd, 0);
        /* read from the shared memory object */
        printf("%s",(char *)ptr);
        /* remove the shared memory object */
        shm_unlink(name);
        return 0;
}
```

Client/Server Sistemlerde Haberleşme

- Client/Server, *bir* ***kaynağın*** ya da bir servisin sağlayıcıları (**server** olarak adlandırılırlar) ile servis istemcileri (**clients**) arasında görevleri bölen bir dağıtık uygulama (distributed application) **mimari stilidir**.
- Client/Server, mimari stilinin **en basit formu** çoklu clientler tarafından direkt erişilebilen *bir server (sunucu) uygulaması içerir*. Bu mimari stili 2-Tier olarak da bilinmektedir.
- Geçmişte client/server mimari stili bir **grafiksel ara yüz** ile veri tabanı ya da **dosya sunucusu** arasında iletişimi sağlamak için kullanılmıştır.

Client/Server Sistemlerde Haberleşme

- **4 tip** haberleşmeden söz etmek mümkündür:
  1. Soketler
  2. Remote Procedure Calls (RPC)
  3. Pipelar
  4. Remote Method Invocation (Java)

Client/Server Soketler

Port

- Bir **bilgisayarla iletişime geçmek** için kullanılan kapılar olarak tanımlanabilir. Portları kullanarak ağdaki belirtilen port ile istenilen bilgisayara ulaşılabilir.

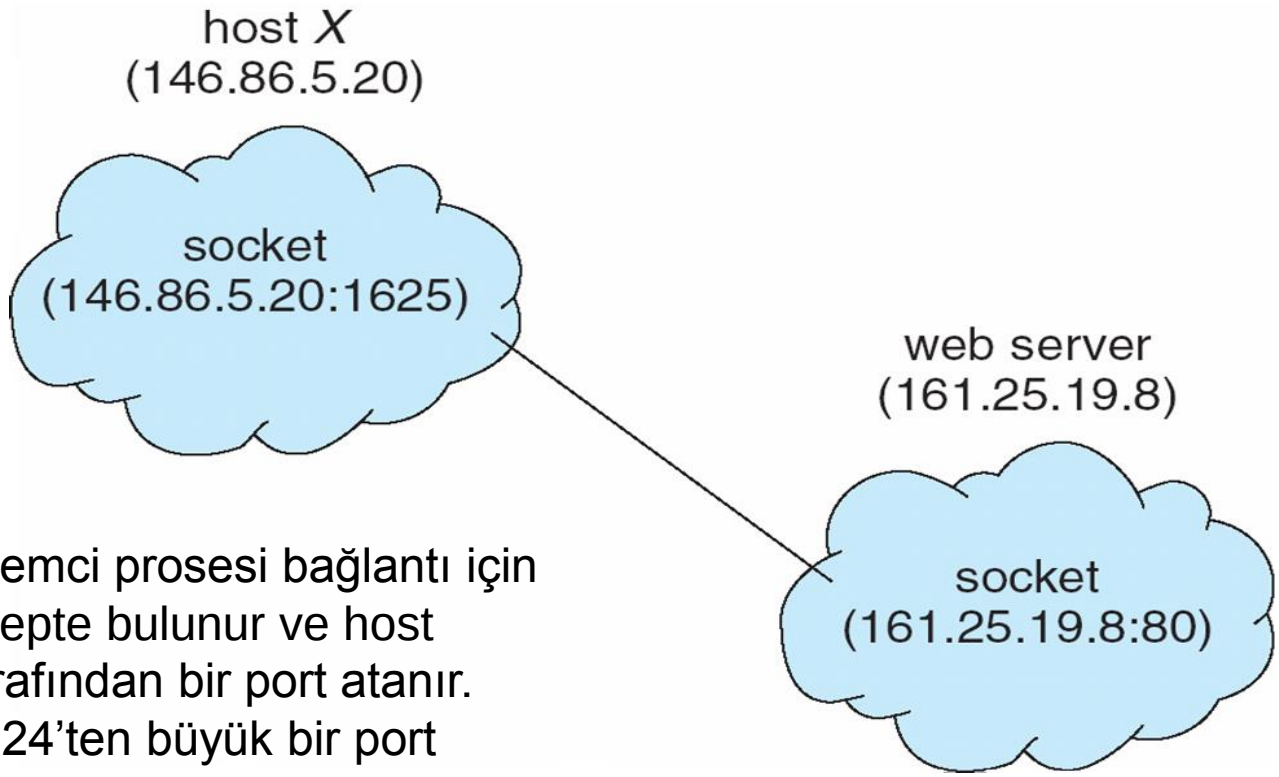
Soket

- Soket, *TCP/IP*'de veri iletişimi için gereken *IP* adresi ve *port* numarasının yan yana yazılması ile oluşan **iletişim kanalıdır**. Örneğin, 192.168.1.1 adresindeki makinaya 23 numaralı porttan yapılacak bağlantı şu şekilde gösterilir:

  [IP adresi] : [port numarası] → **192.168.1.1: 23**

- Bu işleme "**soket açma**" ya da "**soket dinleme**" denilmektedir.

Client/Server Soketler (devam…)



- İstemci prosesi bağlantı için talepte bulunur ve host tarafından bir port atanır.
- 1024'ten büyük bir port numarası atanır.

Client/Server Soket – C# Sunucu

```
static void Main(string[] args)
{
  Socket sck = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
                          ProtocolType.Tcp);
  sck.Bind(new IPEndPoint(0, 1234));
  Console.WriteLine("********SUNUCU************");
  while (true)
  {
      sck.Listen(100);

      Socket soketi_kabul_et = sck.Accept();
      byte[] tampon = new byte[soketi_kabul_et.SendBufferSize];
      int okunan_byte = soketi_kabul_et.Receive(tampon);
      byte[] gelen_mesaj = new byte[okunan_byte];

      for (int i = 0; i < okunan_byte; i++)
      {
        gelen_mesaj[i] = tampon[i];//tampondaki bilgi mesaja atanır.
      }

      string mesaj = Encoding.ASCII.GetString(gelen_mesaj);
      Console.WriteLine("Gelen Mesaj:" + mesaj);
      soketi_kabul_et.Close();
  }
  Console.Read();
  sck.Close();
}
```

Client/Server Soket – C# İstemci

```
Socket sck = new Socket(AddressFamily.InterNetwork,
                        SocketType.Stream,
                        ProtocolType.Tcp);

IPEndPoint yerel = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 1234);

Console.WriteLine("********İSTEMCİ*************");
try
{
    sck.Connect(yerel);
    Console.WriteLine("Sunucuya bağlandı...");
}
catch
{
    Console.WriteLine("Bağlanamadı...");
    Main(args);
}

Console.Write("Mesajınız: ");
string mesaj = Console.ReadLine();

byte[] mesajAsciiHali = Encoding.ASCII.GetBytes(mesaj);
sck.Send(mesajAsciiHali);

Console.WriteLine("Mesaj İletildi");
Console.Read();
sck.Close();
```

Client/Server Pipelar

- Pipe (**tünel**), iki prosesin haberleşmesinde bir kanal olarak kullanılır. Bazı limitleri olsa da iki prosesin haberleşmesindeki ***en basit yöntemlerden birisidir***.
- İki tip pipe bulunmaktadır:
  - **Sıradan** (**Ordinary**) Pipe
  - **Adlandırılmış** (**Named**) Pipe
- **Sorular**
  - İletişim tek yönlü mü çift yönlü mü?
  - İletişim kuran prosesler arasında parent-child ilişkisi olmak zorunda mı?
  - Pipe'lar network üzerinde de kullanılabilir.

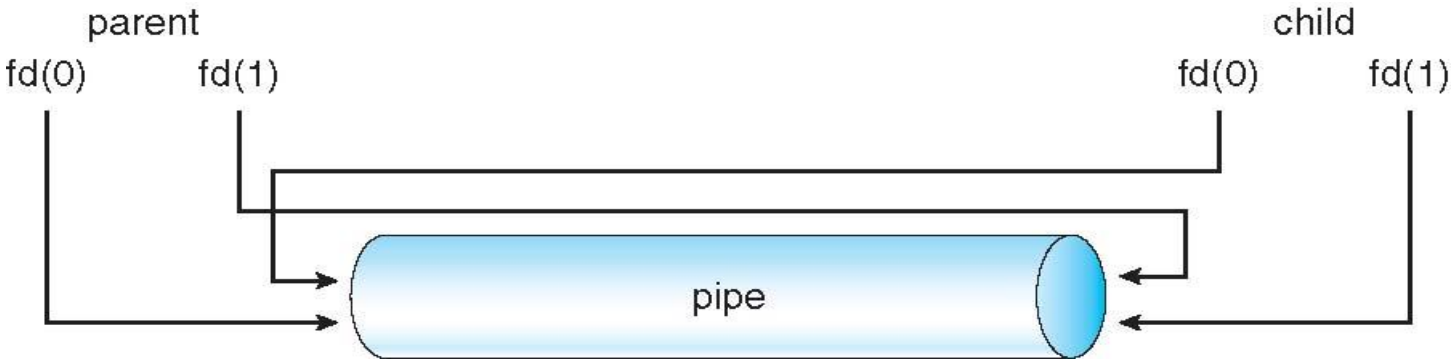
Ordinary Pipe

- Sıradan tüneller, **üretici-tüketici** tipi iletişime izin verir.
- Producer, *bir uçtan yazar* (tünelin yazma ucu)
- Consumer, *diğer ucundan okur* (tünelin okuma ucu)
- Sıradan tüneller bu nedenle **tek yönlü iletişim** sağlar.
- Haberleşen prosesler arasında **parent-child ilişkisi gereklidir**.
- Genelde, **parent proses** pipe'ı yaratır ve fork() kullanarak yarattığı **child ile iletişim için kullanır**.
- Child proses, *parent tarafından açılmış dosyaları* **miras aldığı** ve **pipe da özel bir çeşit dosya olduğu için** pipe'lar da miras alınabilir.
- Windows da bu pipe'lar **anonymous pipes** olarak geçer.

Ordinary Pipe (devam…)

```
int pipe_fds[2];
int read_fd;
int write_fd;
pipe(pipe_fds);
read_fd = pipe_fds[0];
write_fd = pipe_fds[1];
```

- write_fd'ye yazılan veri, read_fd ucundan okunabilir.



Named Pipe

- Adlandırılmış pipe'lar, sıradan olanlardan daha güçlüdür.
- İletişim **çift yönlüdür**.
- Haberleşen prosesler arasında **parent-child** ilişkisi gerekli değildir.
- Birden fazla proses, kullanabilir.
- Birden fazla proses yazabilir, tek proses okur. Yazma işlemi **atomiktir**.
- Pipe'lar message passing olarak kullanılırlar.
- UNIX ve Windows işletim sistemlerince desteklenir.
- UNIX işletim sistemlerinde **FIFOs** olarak geçer.
- FIFO yaratmak için **mkfifo()** sistem çağrısı kullanılır.

Araştırma Soruları

- Message Passing vs Shared Memory
- Shared Memory vs Named Pipe
- Why Named pipe instead of file?

İYİ ÇALIŞMALAR…

Yararlanılan Kaynaklar

- **Ders Kitabı:**
  - Operating System Concepts, Ninth Edition, Abraham Silberschatz, Peter Bear Galvin, Greg Gagne
- **Yardımcı Okumalar:**
  - İşletim Sistemleri, Ali Saatçi
  - Şirin Karadeniz, Ders Notları
  - İbrahim Türkoğlu, Ders Notları