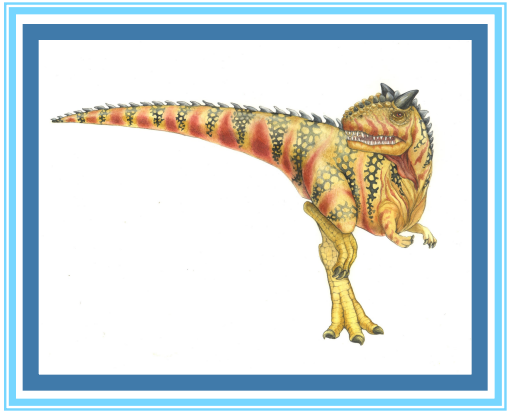
Chapter 3_2: Processes

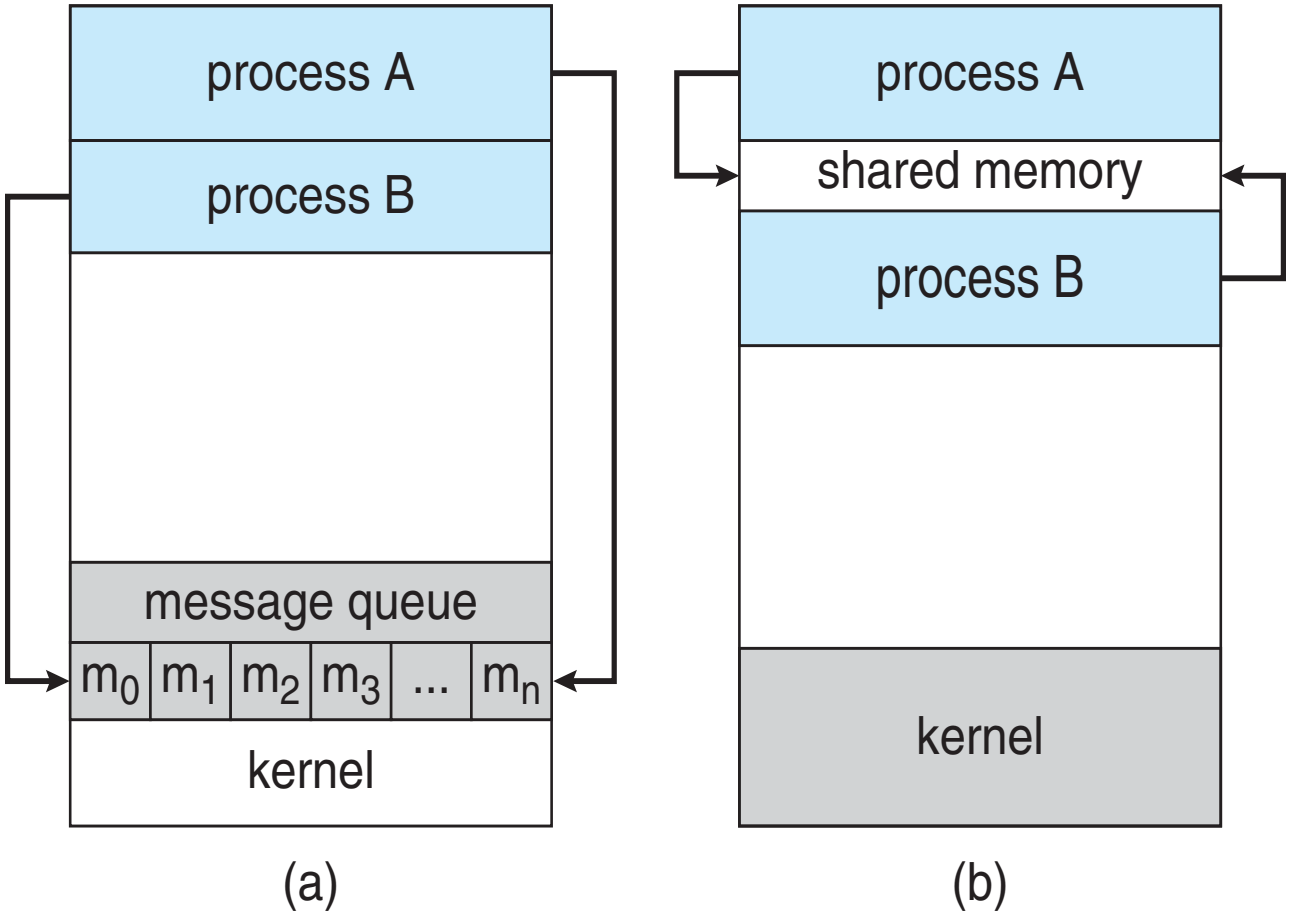


Interprocess Communication

- Processes within a system may be ***independent*** or ***cooperating***
- Cooperating process can affect or be affected by other processes, including sharing data (İşlemler bağımsız çalışabilir ya da beraber)
- Reasons for cooperating processes: (İşbirliği sebebi ne olabilir?)
  - Information sharing(Bilgi paylaşımı)
  - Computation speedup(Hesaplama hızı)
  - Modularity(modülerlik)
  - Convenience(Kolaylık)
- Cooperating processes need **interprocess communication** (**IPC**)
- Two models of IPC
  - **Shared memory (Paylaşımlı Bellek)**
  - **Message passing(Mesaj geçirme)**

Communications Models

(a) Message passing. (b) shared memory.



Cooperating Processes

- ***Independent*** process cannot affect or be affected by the execution of another process
- ***Cooperating*** process can affect or be affected by the execution of another process
- Advantages of process cooperation
  - Information sharing
  - Computation speed-up
  - Modmodularity
  - Convenience

Bağımsız işlemler diğerlerine etki etmezler.

Producer-Consumer Problem

- Paradigm for cooperating processes, *producer* process produces information that is consumed by a *consumer* process
  - **unbounded-buffer** places no practical limit on the size of the buffer
  - **bounded-buffer** assumes that there is a fixed buffer size

- Üretici ve tüketicinin bir arada çalışabilmesi için buffer havuzları oluşturmalıyız.
- Bu havuzları üretici doldurmalı , tüketici boşaltmalı. Bu sayede üretici , tüketici diğer taraftan havuzu boşaltırken havuzu doldurabilmeli.
- Üretici proses bir veriyi tüketici proses için üretir ve tüketici proses bu veriyi kullanır.

Bounded-Buffer – Shared-Memory Solution

- Shared data

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

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

- Solution is correct, but can only use BUFFER_SIZE-1 elements

Bounded-Buffer – Producer

```
item next_produced;
while (true) {
        /* produce an item in next produced */
        while (((in + 1) % BUFFER_SIZE) == out)
                ; /* do nothing */
        buffer[in] = next_produced;
        in = (in + 1) % BUFFER_SIZE;
}
```

Bounded Buffer – Consumer

```
item next_consumed;

while (true) {
        while (in == out)
                ; /* do nothing */
        next_consumed = buffer[out];

        out = (out + 1) % BUFFER_SIZE;

        /* consume the item in next consumed */

}
```

- Üretici ve tüketicinin *sınırlı* bir veri alanına (buffer) erişebildiklerini varsayalım.
- Üretici üretmiş olduğu ürünü (veriyi) Buffer'da boş yer varsa koyabilir. Yer yok ise tüketicinin Buffer'dan bir veriyi almasını beklemek durumundadır.
- Tüketici ise ancak Buffer boş değilse veri tüketebilir, aksi taktirde üreticinin Buffer'a veri girmesini beklemesi gerekir.
- Unbounded (sınırlanmamış) – buffer üretici tüketici probleminde bufferların sayısında bir sınır yoktur. Tüketici yeni ürünler bekler , ve gelenleri tüketir , sadece ürün bitince bekler fakat üretici daima yeni ürün üretip koyabilir. Bounded producer / consumer probleminde ise n adet sabit buffer vardır. Üretici n buffer dolunca bekler , tüketici n adet buffer boşalınca bekler

Interprocess Communication – Shared Memory

- An area of memory shared among the processes that wish to communicate
- The communication is under the control of the users processes not the operating system.
- Major issues is to provide mechanism that will allow the user processes to synchronize their actions when they access shared memory.
- Synchronization is discussed in great details in Chapter 5.

Problem: Kullanıcı işlemlerinin paylaşılan belleğe eriştiklerinde işlemlerini senkronize etmesine olanak tanıyacak mekanizma sağlama işlemi(Kim hangi anda kullanıcak)

Interprocess Communication – Message Passing

- Mechanism for processes to communicate and to synchronize their actions
- Message system – processes communicate with each other without resorting to shared variables
- IPC facility provides two operations:
  - **`send`**(*message*)
  - **`receive`**(*message*)
- The *message* size is either fixed or variable

Message Passing (Cont.)

- If processes *P* and *Q* wish to communicate, they need to:
  - Establish a ***communication link*** between them
  - Exchange messages via send/receive
- Implementation issues:
  - 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?

Gerçekleştirim Soruları

- Bağlantılar nasıl sağlanır?
- Bir bağlantı ikiden fazla işlemle ilişkilendirilmeli midir?
- İletişim kuran işlemlerin her bir çifti arasında kaç tane bağlantı olabilir?
- Bağlantının kapasitesi nedir?
- Bağlantının desteklediği mesajların boyutu sabit mi yoksa değişken midir?
- Bağlantı çift yönlü mü (bi-directional) yoksa tek yönlü (unidirectional) müdür?

Message Passing (Cont.)

- Implementation of communication link
  - Physical:
    - Shared memory
    - Hardware bus
    - Network
  - Logical:
    - Direct or indirect
    - Synchronous or asynchronous
    - Automatic or explicit buffering

Direct Communication

- Processes must name each other explicitly:
  - **`send`** (*P, message*) – send a message to process P
  - **`receive`**(*Q, message*) – receive a message from process Q
- Properties of communication link
  - Links are established automatically
  - A link is associated with exactly one pair of communicating processes
  - Between each pair there exists exactly one link
  - The link may be unidirectional, but is usually bi-directional

Direk İletişim

- İşlemler birbirlerini açıkça isimlendirmelidir:
  - **send (P, message)** – P işlemine bir mesaj gönder
  - **receive(Q, message)** – Q işleminden bir mesaj al
- İletişim bağlantısının özellikleri
  - Bağlantılar otomatik olarak sağlanır
  - Bir bağlantı sadece bir çift işlemci arasında oluşturulur
  - Her bir çift için sadece bir bağlantı oluşturulur
  - Bağlantı tek yönlü olabilir, ama genellikle çift yonlüdür

Indirect Communication

- Messages are directed and received from mailboxes (also referred to as ports)
  - Each mailbox has a unique id
  - Processes can communicate only if they share a mailbox
- Properties of communication link
  - Link established only if processes share a common mailbox
  - A link may be associated with many processes
  - Each pair of processes may share several communication links
  - Link may be unidirectional or bi-directional

Dolaylı İletişim

- Mesajlar **posta kutusuna (messagebox)** yönlendirilir ve post kutusundan alınır
  - Posta kutusu yerine **port** terimi de kullanılır
  - Her bir posta kutusu özgün bir ada sahiptir (**unique id**)
  - İşlemler sadece bir post kutusunu paylaşıyor ise iletişime geçebilir
- İletişim bağlantısının özellikleri
  - Bağlantı sadece işlemler ortak bir posta kutusunu paylaşıyor ise oluşturulur
  - İkiden fazla işlem tek bir posta kutusu ile ilişkilendirilebilir
  - Bir işlem çifti birden fazla iletişim bağlantısına sahip olabilir
  - Bağlantı tek yönlü veya çift yönlü olabilir

Indirect Communication

- Operations
  - create a new mailbox (port)
  - send and receive messages through mailbox
  - destroy a mailbox
- Primitives are defined as:

  `send`(*A, message*) – send a message to mailbox A

  `receive`(*A, message*) – receive a message from mailbox A

Indirect Communication

- Mailbox sharing
  - P_1, P_2, and P_3 share mailbox A
  - P_1, sends; P_2 and P_3 receive
  - Who gets the message?
- Solutions
  - Allow a link to be associated with at most two processes
  - Allow only one process at a time to execute a receive operation
  - Allow the system to select arbitrarily the receiver. Sender is notified who the receiver was.

Posta Kutusu Paylaşımı

- Posta kutusu paylaşımı
  - P_1, P_2, ve P_3 A posta kutusunu paylaşıyor
  - P_1, meaj gönderir; P_2 *ve* P_3 alır
  - Mesajı kim alır?
- Çözümler
  - Bir bağlantının en fazla iki işlem ile ilişkilendirilmesine izin ver
  - Herhangi bir anda sadece bir işlemin mesaj almasına izin ver
  - Sistemin herhangi bir alıcıyı seçmesine izin ver. Gönderici kimin mesajı aldığı konusunda bilgilendirilir

Buffering

- Queue of messages attached to the link.
- implemented in one of three ways
  1. Zero capacity – no messages are queued on a link.
     Sender must wait for receiver (rendezvous)
  2. Bounded capacity – finite length of *n* messages
     Sender must wait if link full
  3. Unbounded capacity – infinite length
     Sender never waits

Senkronizasyon

- Mesaj gönderme **bloklanan (blocking)** veya **bloklanmayan (non-blocking)** şekilde gerçekleşebilir
- Bloklanan mesajlaşma senkrondur (synchronous)
  - **Bloklanan gönderimde**, alıcı taraf mesajı alana kadar, gönderici taraf bloklanır
  - **Bloklanan alımda**, gönderici taraf mesajı gönderene kadar, alıcı taraf bloklanır
- Bloklanmayan mesajlaşma asenkrondur (asynchronous)
  - **Bloklanmayan gönderimde**, gönderici taraf mesajı gönderdikten sonra beklemeksizin çalışmaya devam eder
  - **Bloklanmayan alımda**, alıcı taraf ya geçerli bir mesaj alır ya da null mesaj alarak çalışmaya devam eder

Examples of IPC Systems - POSIX

- POSIX Shared Memory
  - Process first creates shared memory segment
    ```
    shm_fd = shm_open(name, O CREAT | O RDWR, 0666);
    ```
  - Also used to open an existing segment to share it
  - Set the size of the object
    ```
    ftruncate(shm fd, 4096);
    ```
  - Now the process could write to the shared memory
    ```
    sprintf(shared memory, "Writing to shared
    memory");
    ```

IPC Sistem Örnekleri - POSIX

- POSIX Paylaşımlı Hafıza
  - İşlem öncelikle paylaşımlı hafıza segmentini oluşturur

```
segment id = shmget(IPC PRIVATE, size, S IRUSR | S
  IWUSR);
```

  - Paylaşımlı belleğe erişmek isteyen işlem, bu paylaşım alanı ile bağlantı kurmalıdır

```
shared memory = (char *) shmat(id, NULL, 0);
```

  - Şimdi işlem paylaşımlı hafızaya yazabilir

```
sprintf(shared memory, "Writing to shared memory");
```

  - Paylaşım alanı ile yapacak işi kalmayan işlem,bu paylaşım alanı ile bağlantısını sonlandırmalıdır

```
shmdt(shared memory);
```

IPC POSIX Producer

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <sys/shm.h>
#include <sys/stat.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);
```

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

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

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 Consumer

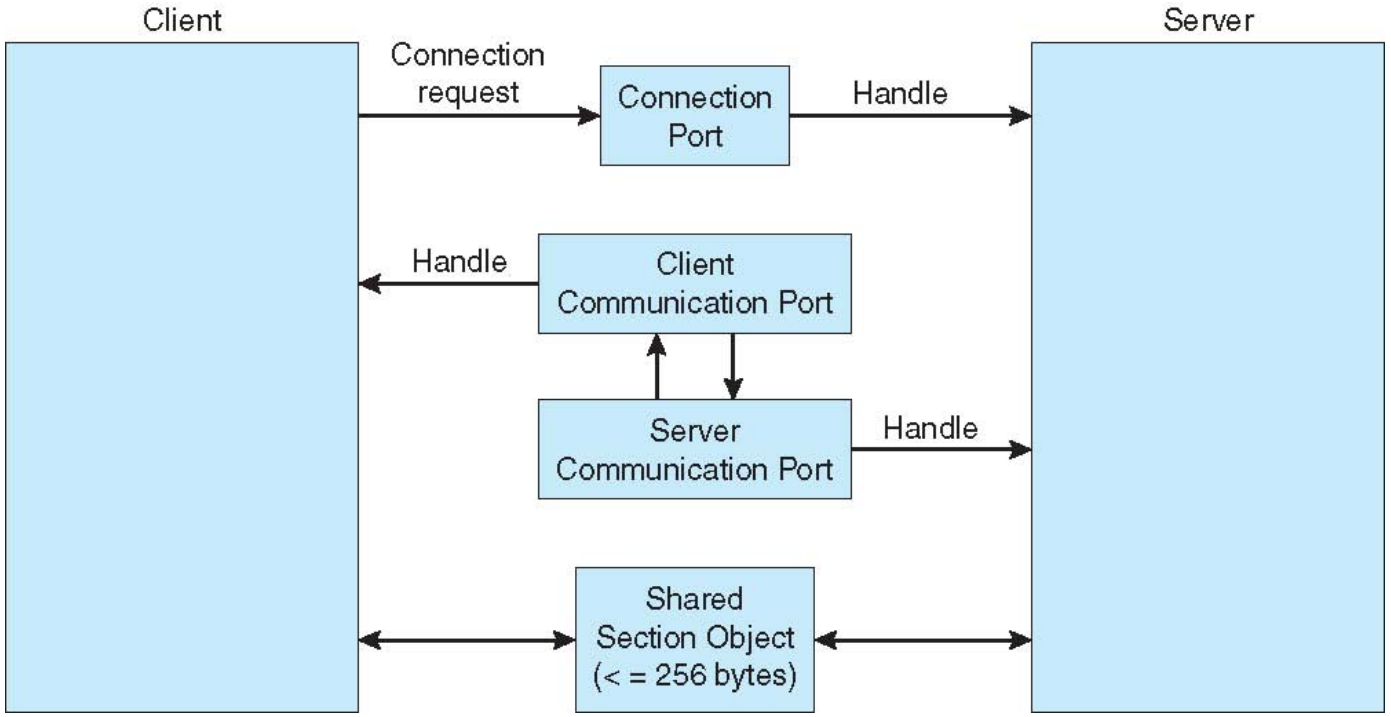
```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/shm.h>
#include <sys/stat.h>

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;
}
```

shm_unlink: Okunan segmenti siler

Examples of IPC Systems – Windows

- Message-passing centric via **advanced local procedure call (LPC)** facility
  - Only works between processes on the same system
  - Uses ports (like mailboxes) to establish and maintain communication channels
  - Communication works as follows:
    - The client opens a handle to the subsystem's **connection port** object.
    - The client sends a connection request.
    - The server creates two private **communication ports** and returns the handle to one of them to the client.
    - The client and server use the corresponding port handle to send messages or callbacks and to listen for replies

Local Procedure Calls in Windows



1.Küçük mesajlar (256 bayt'a kadar) için, portun mesaj sırası kullanılır(ara depolama alanı)
di
2. Daha büyük mesajlar bir bölüm nesnesinden geçirilmelidir. Kanalla ilişkili paylaşılan hafıza bölgesi.
3. Verilerin miktarı bir bölüm nesnesine sığmayacak kadar büyük olduğunda, bir API sunucu işlemlerinin doğrudan okuma ve yazma izni ile çalışır.

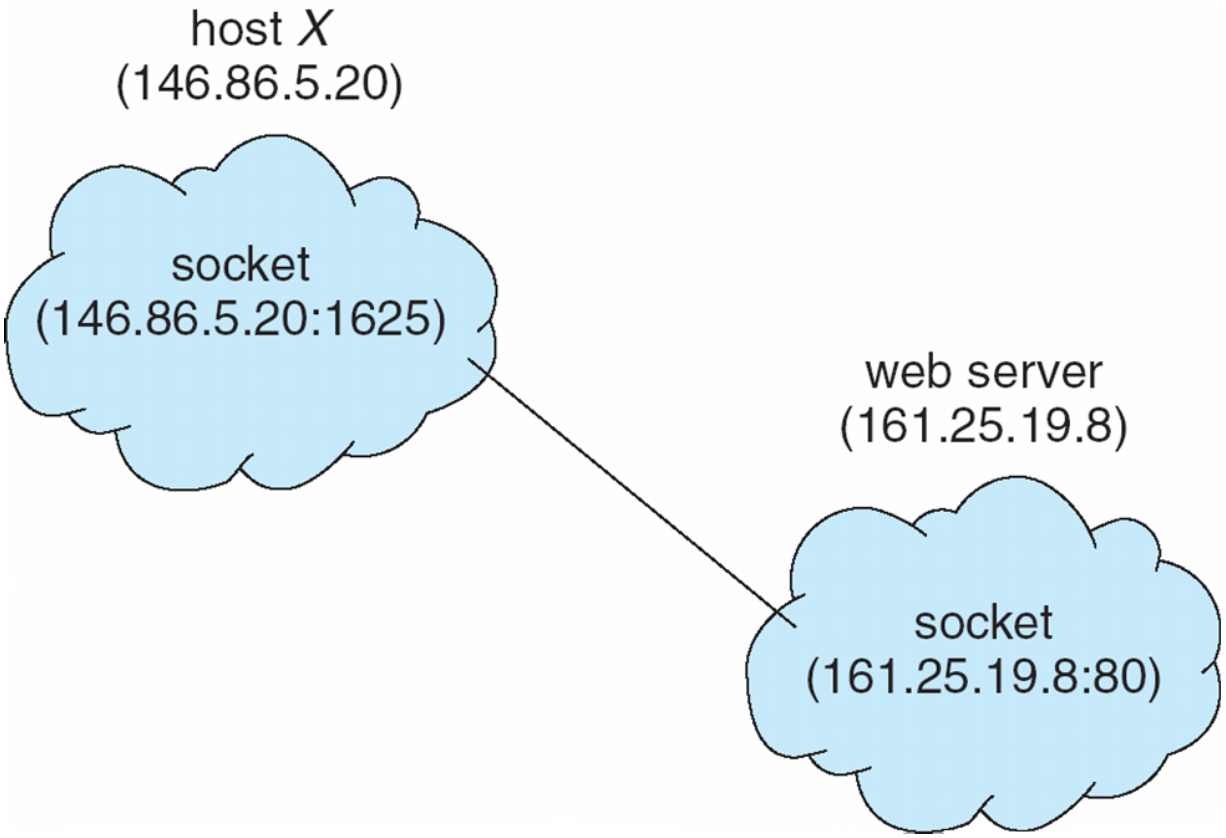
Communications in Client-Server Systems

- Sockets
- Remote Procedure Calls
- Pipes
- Remote Method Invocation (Java)

Sockets

- A **socket** is defined as an endpoint for communication
- Concatenation of IP address and **port** – a number included at start of message packet to differentiate network services on a host
- The socket **161.25.19.8:1625** refers to port **1625** on host **161.25.19.8**
- Communication consists between a pair of sockets
- All ports below 1024 are ***well known***, used for standard services
- Special IP address 127.0.0.1 (**loopback**) to refer to system on which process is running

Socket Communication



Sockets in Java

- Three types of sockets
  - **Connection-oriented** (**TCP**)
  - **Connectionless** (**UDP**)
  - **`MulticastSocket`** class– data can be sent to multiple recipients
- Consider this “Date” server:

```
import java.net.*;
import java.io.*;

public class DateServer
{
   public static void main(String[] args) {
      try {
         ServerSocket sock = new ServerSocket(6013);

         /* now listen for connections */
         while (true) {
            Socket client = sock.accept();

            PrintWriter pout = new
             PrintWriter(client.getOutputStream(), true);

            /* write the Date to the socket */
            pout.println(new java.util.Date().toString());

            /* close the socket and resume */
            /* listening for connections */
            client.close();
         }
      }
      catch (IOException ioe) {
         System.err.println(ioe);
      }
   }
}
```

Remote Procedure Calls

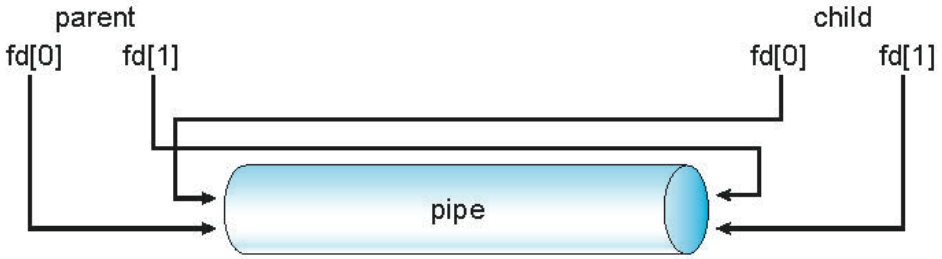
- Remote procedure call (RPC) abstracts procedure calls between processes on networked systems
  - Again uses ports for service differentiation
- **Stubs** – client-side proxy for the actual procedure on the server
- The client-side stub locates the server and **marshalls** the parameters
- The server-side stub receives this message, unpacks the marshalled parameters, and performs the procedure on the server
- On Windows, stub code compile from specification written in **Microsoft Interface Definition Language** (**MIDL**)
- (Uzak Yordam çağrısı- mail server gibi)
- Bağlantı kurmak için servis

Pipes

- Acts as a conduit allowing two processes to communicate
- Issues:
  - Is communication unidirectional or bidirectional?
  - In the case of two-way communication, is it half or full-duplex?
  - Must there exist a relationship (i.e., ***parent-child***) between the communicating processes?
  - Can the pipes be used over a network?
- Ordinary pipes – cannot be accessed from outside the process that created it. Typically, a parent process creates a pipe and uses it to communicate with a child process that it created.
- Named pipes – can be accessed without a parent-child relationship.
- Pipe (tünel), iki prosesin haberleşmesinde bir kanal olarak kullanılır.
- • İ Sıradan (Ordinary) Pipe – Adlandırılmış (Named) Pipe
- İletişim tek yönlü mü çift yönlü mü?
- – İletişim kuran prosesler arasında parent-child ilişkisi olmak zorunda mı?
- – Pipe'lar network üzerinde de kullanılabilir mi?

Ordinary Pipes

- Ordinary Pipes allow communication in standard producer-consumer style
- Producer writes to one end (the **write-end** of the pipe)
- Consumer reads from the other end (the **read-end** of the pipe)
- Ordinary pipes are therefore unidirectional
- Require parent-child relationship between communicating processes
- Windows calls these **anonymous pipes**



Sıradan tüneller, üretici-tüketici tipi iletişime izin verir.

- Producer, bir uçtan yazar
- 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

Named Pipes

- Named Pipes are more powerful than ordinary pipes
- Communication is bidirectional
- No parent-child relationship is necessary between the communicating processes
- Several processes can use the named pipe for communication
- Provided on both UNIX and Windows systems

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.
- Pipe'lar message passing olarak kullanılırlar.
- UNIX ve Windows işletim sistemlerince desteklenir.

End of Chapter 3