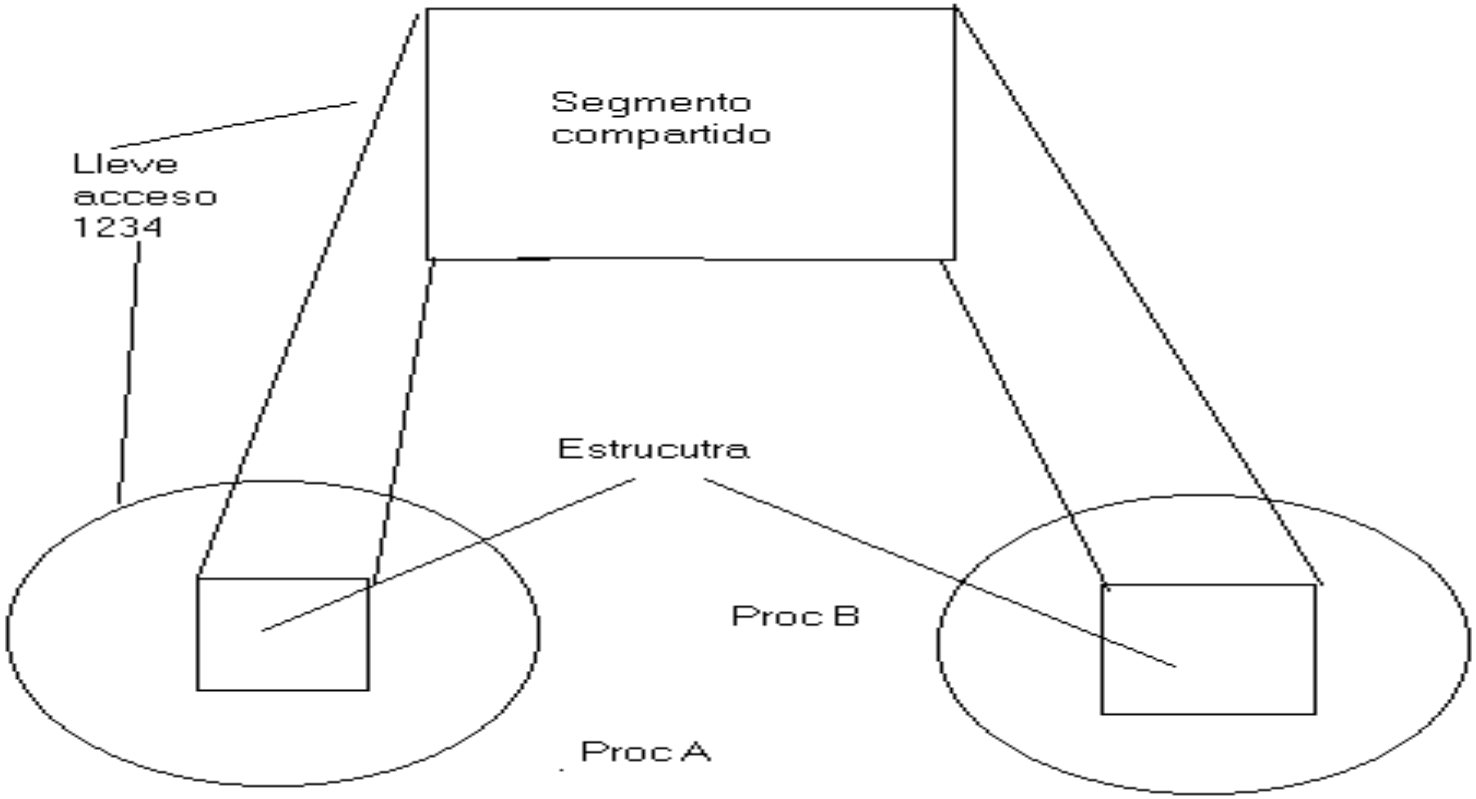
IPC - Inter Process Communication

Memoria – Pipes – Sockets –
Rendez Vous - Mailbox

Memoria Compartida



Memoria Compartida

- Bibliotecas
- #include <sys/types.h>
- #include <sys/ipc.h>
- #include <sys/shm.h>

- Struct shmid_ds {
  - struct ipc_perm Shm_perm; (permisos)
  - int shm_segz (tamaño segmento)
  - struct XXX shm_YYY (depende de la info)
  - ushort shm_cpid (creador pid)
  - Timepos ...

Memoria Compartida (pasos)

- Obtener manejador de segmentos
- “Atar” seg. a una estructura de memoria
- Leer/Escribir
- “Desatar”
- Borrar

Memoria Compartida

- id =shmget(key, size, flag)

- key = ftok (file_name, id_key)
- size = tamaño
- flag = permisos

- int id (manejador de segmento)

Memoria Compartida

- ptr = shmat(id, addr, flag)
- (ata un segmento en el espacio de direcciones del proceso)
- Ejemplo:

  Struct info {int cant_lec; int cant_escr; } ;

  Struct info *ptr;

  ptr = (struct info *) shmat(id, 0, 0)
- Uso la memoria compartida con *ptr.cant_lec

Memoria Compartida

- shmdt(addr) desata sin borrar
- shmctl(id, cmd, buf)
  - id manejador de segmento
  - cmd comando
  - buf almacena info del estado

Memoria Compartida (secuencia)

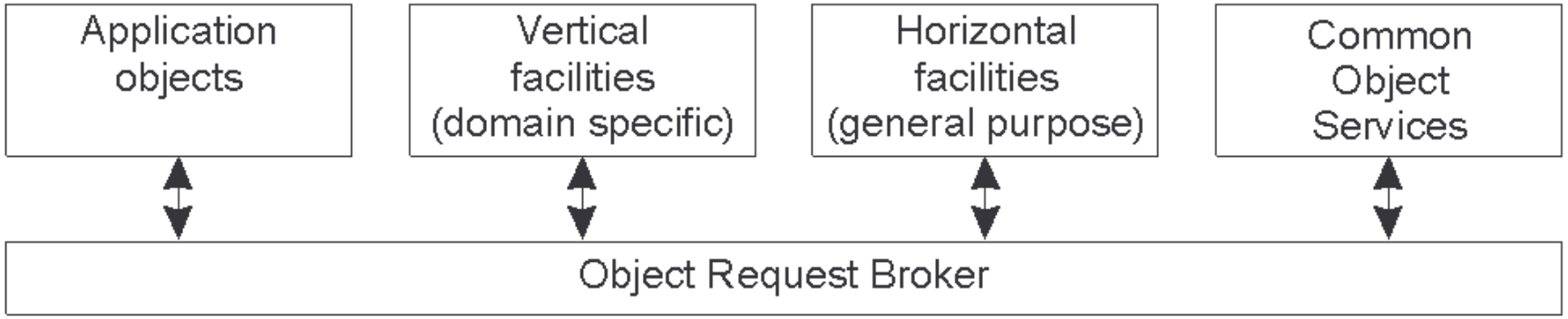
- struct info {int cant_escr} ;
- struct info *ptr
- id = shmget(key,size, ...)
- ptr = shmat(id, 0, 0);
- *ptr.cant_escr ... (uso)
- shmdt(ptr)
- shmctl(id,IPC_RMID,0)

DSHM (Memoria compartida distribuida)

- Basado en páginas
- Mapeador de Páginas
- Primitivas similares a lo anterior
- El pedido de un página no existente genera page-fault para traer la página
- Mecanismos de consistencia

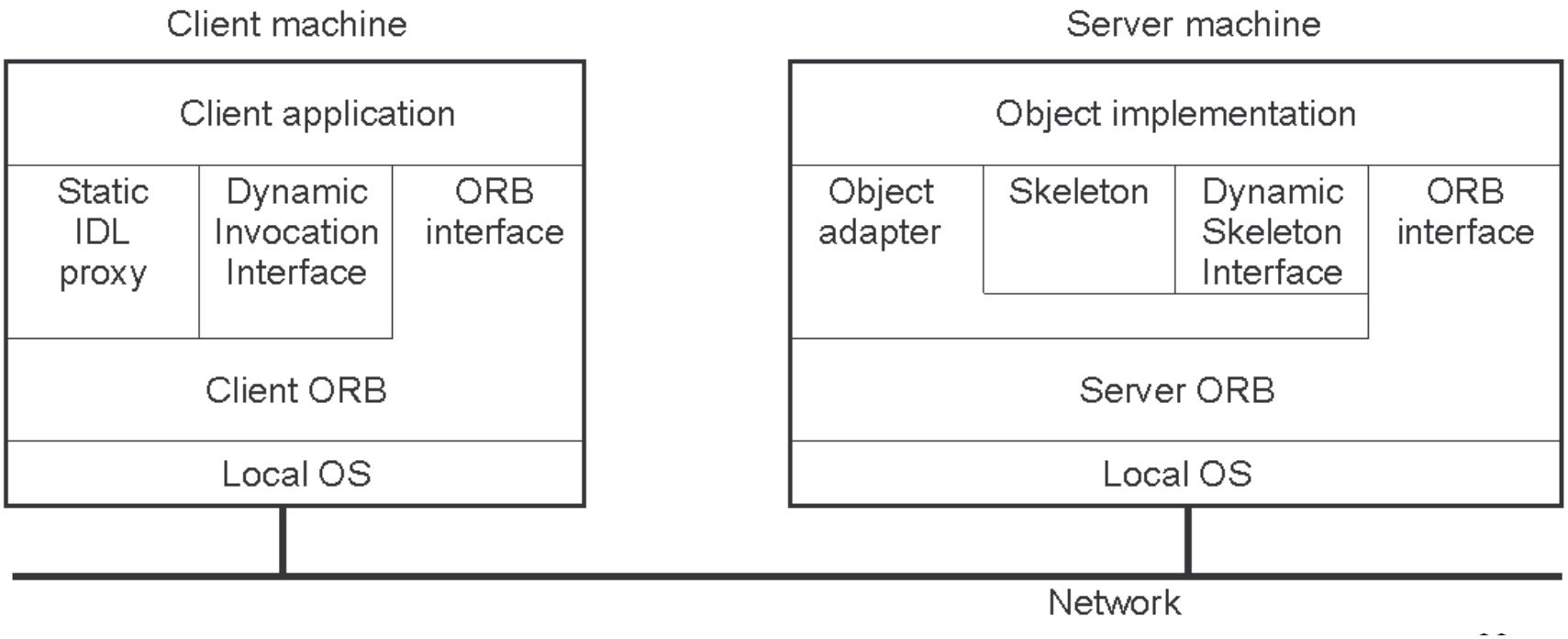
DSHM (Memoria comparida distribuida)

- Basada en Objetos (CORBA) (fijos)



DSHM (Memoria comparida distribuida)

- Modelo

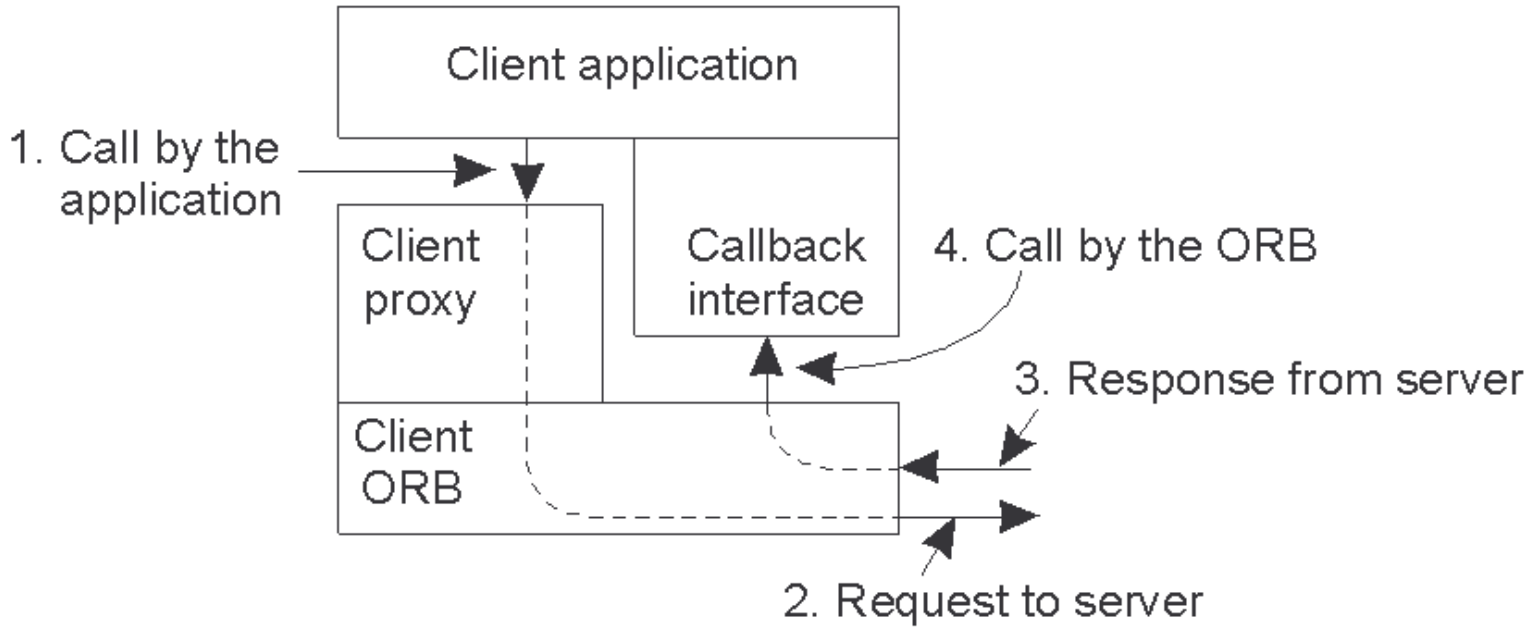


DSHM (Memoria comparida distribuida)

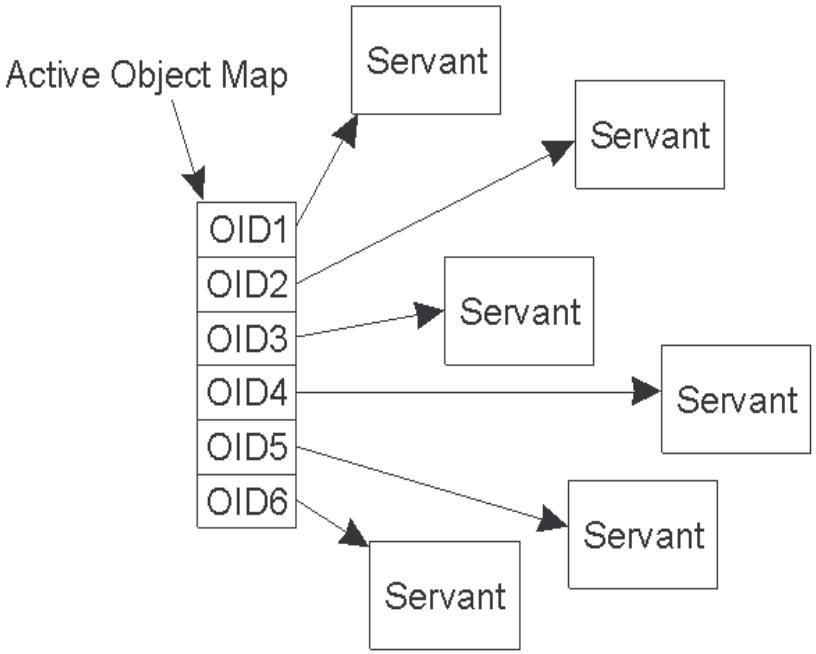
- Modelos

Request type	Failure semantics	Description
Synchronous	At-most-once	Caller blocks until a response is returned or an exception is raised
One-way	Best effort delivery	Caller continues immediately without waiting for any response from the server
Deferred synchronous	At-most-once	Caller continues immediately and can later block until response is delivered

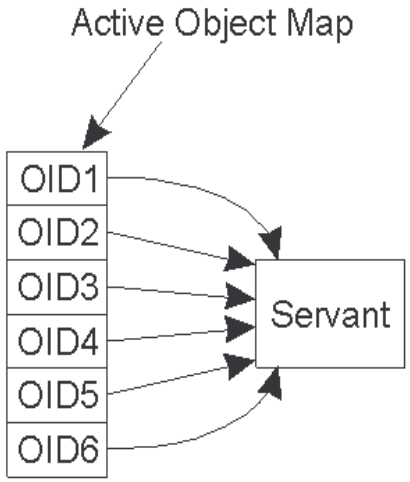
DSHM (Memoria comparida distribuida)



DSHM (Memoria comparida distribuida)



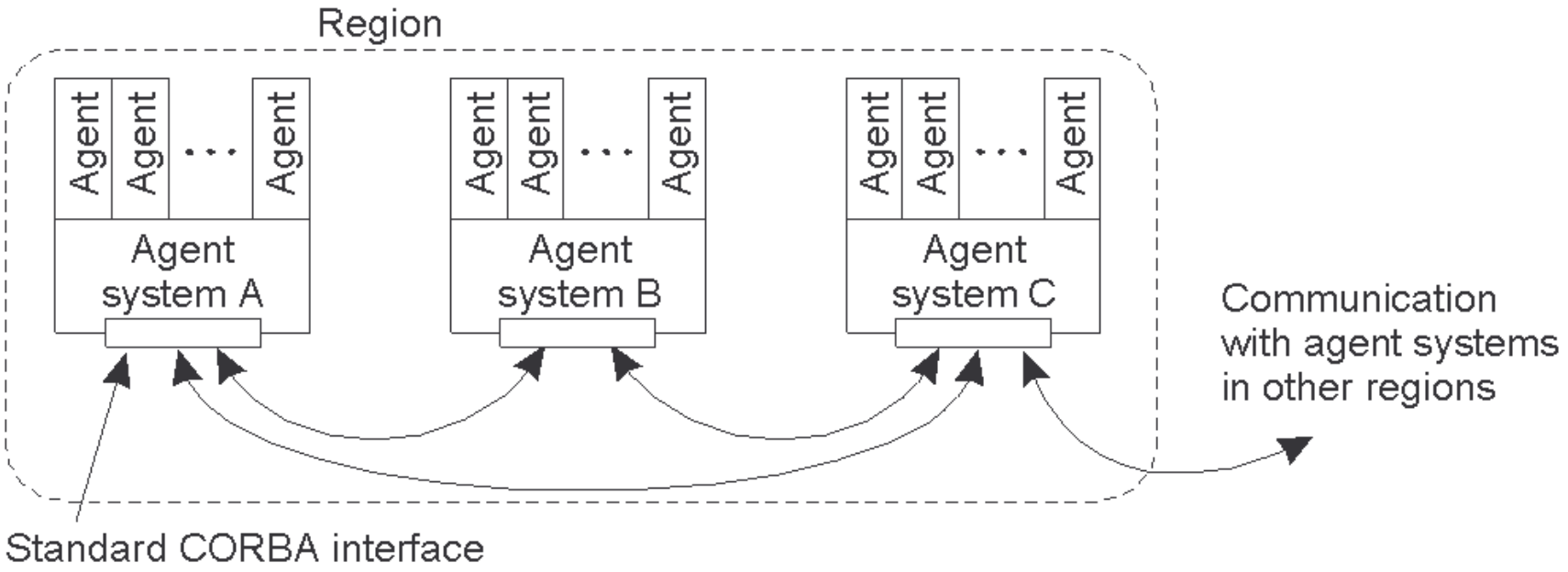
(a)



(b)

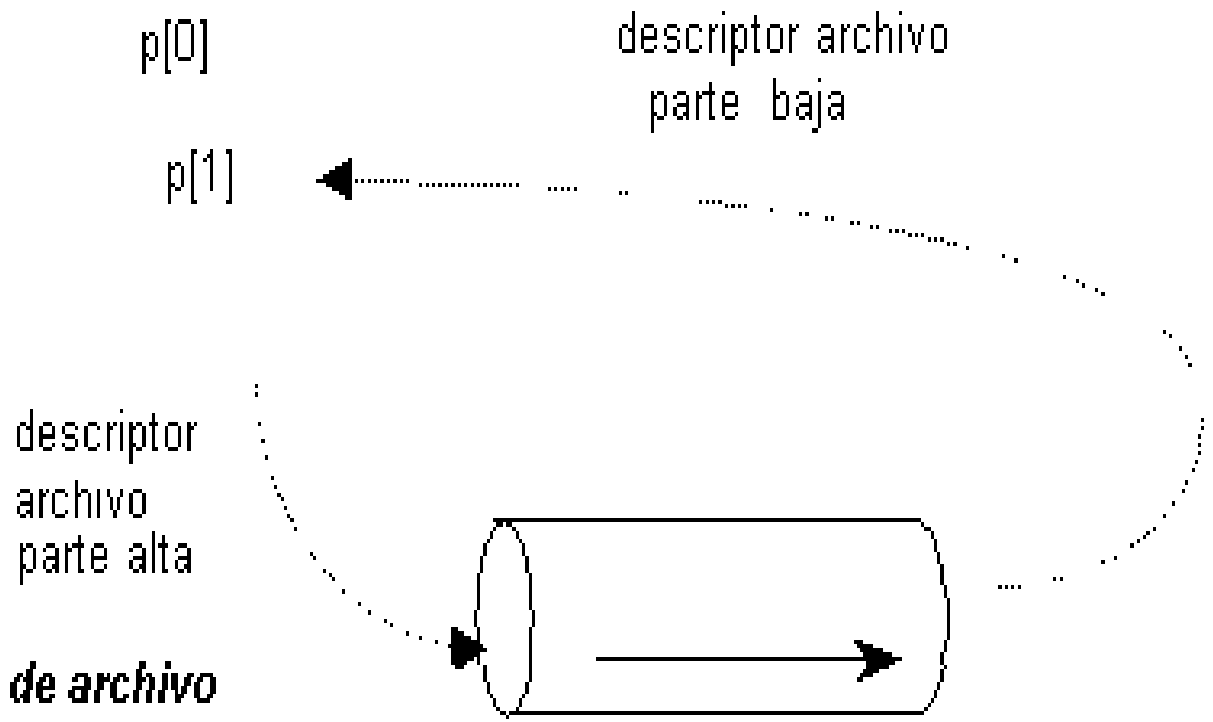
DSHM (Memoria comparida distribuida)

- Agentes (móbiles)



Pipes

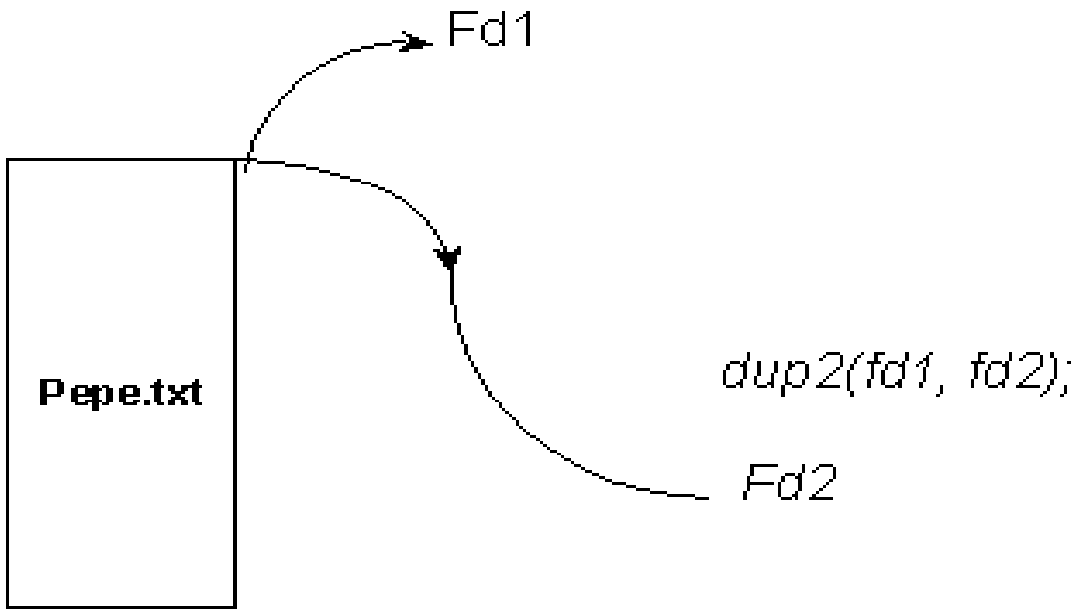
- ***int p(2);***
  pipe(p);



descriptor de archivo parte alta

Pipes

- ***#include <unistd.h>***
- ***int dup2(int fd1, int fd2);***



Pipes (who | more)

```
/* Archivo del programa whomore.c  */
main()
{
      int fds[2]
      pipe(fds);
      /* Hijo1 reconecta stdin a  parte baja del pipe y cierra alta */
      if (fork() == 0)  {
        dup2(fds[0], 0);
        close(fds[1]);
        execlp("more","more",0);
     }
     else {
       /* Hijo2 reconecta stdout  a  parte alta del pipe y cierra baja */
        if ( fork() == 0 )          {
       dup2(fds[1], 1);
       close(fds[0]);
       execlp("who", "who", 0);
        }
        else {    /* padre cierra ambas partes y espera a los hijos */
       close(fds[0]);
       close(fds[1]);
       wait(0);
       wait(0):
        }
}
```

Pipes

- Las limitaciones de los pipes residen en:

1. Pipes son unidireccionales. La solución para lograr comunicación en dos sentidos es crear dos pipes.
2. Pipes no pueden autentificar al proceso con el que mantiene comunicación.
3. Pipes deben de ser pre-arreglados. Dos procesos no relacionados no pueden conectarse vía pipes, deben tener un ancestro común que cree el pipe y se los herede.
4. Pipes no trabajan a través de una red. Los dos procesos deben encontrarse en la misma máquina.

Sockets

1º) El proceso servidor crea un socket con nombre y espera la conexión.

2º) El proceso cliente crea un socket sin nombre.

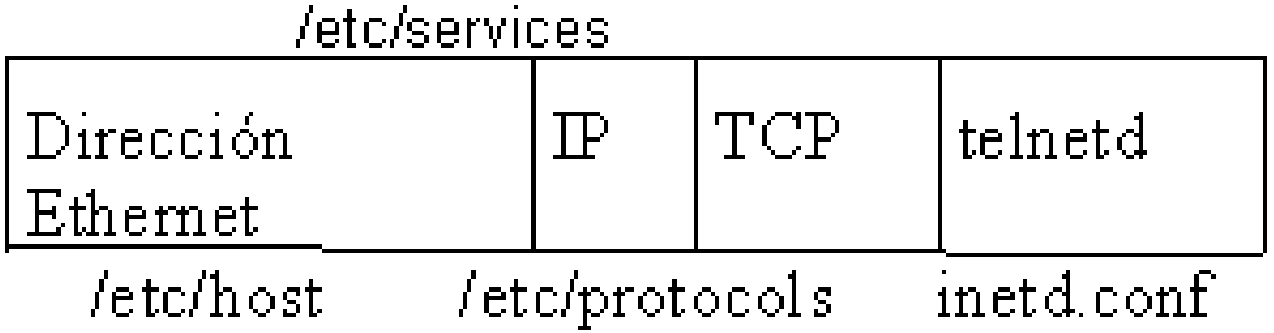
3º) El proceso cliente realiza una petición de conexión al socket servidor.

4º) El cliente realiza la conexión a través de su socket mientras el proceso servidor mantiene el socket servidor original con nombre.

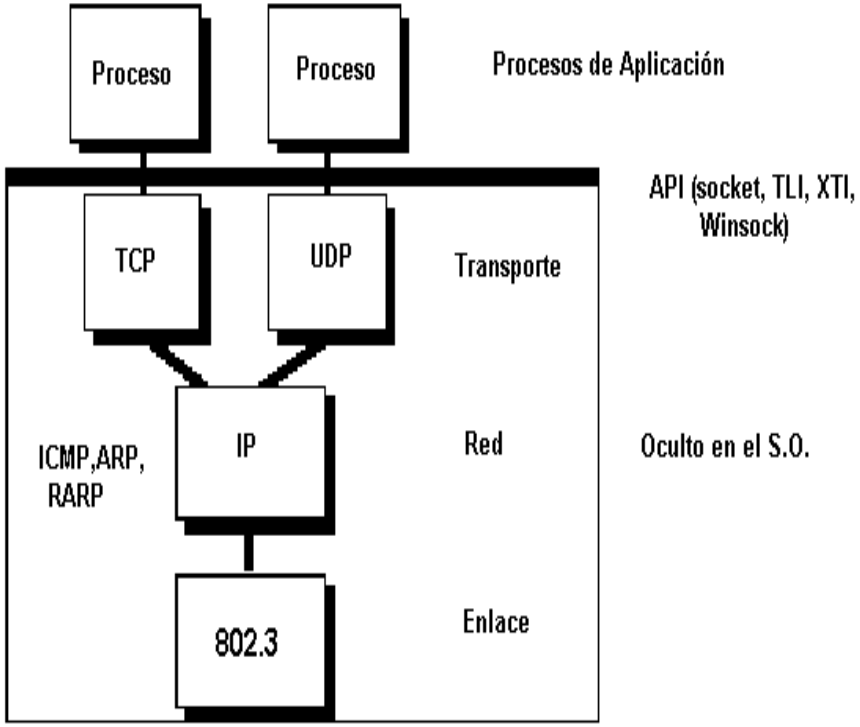
Modelo TCP/IP

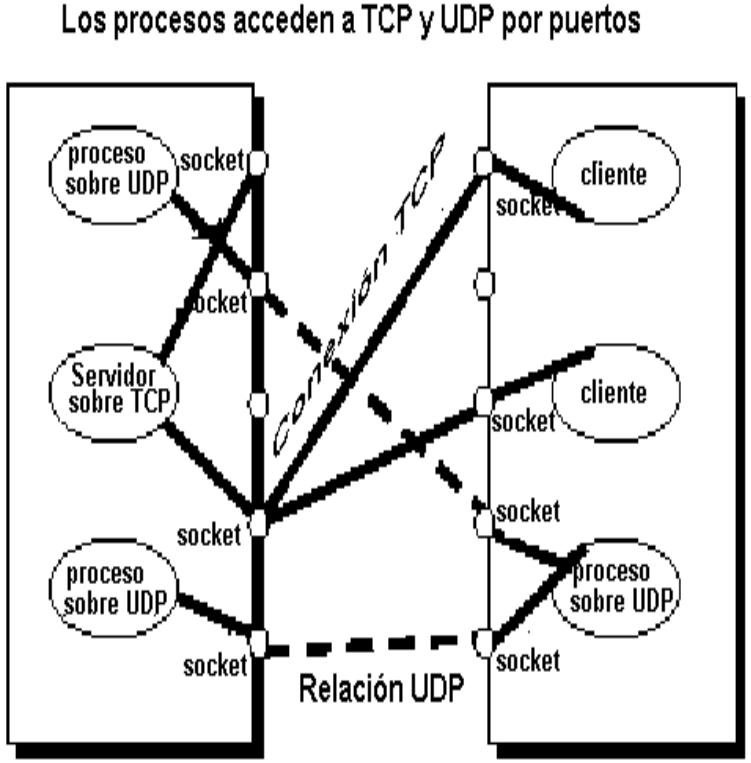
	TCP/IP	OSI
(NFS)		7. Aplicación
(XDR)	5. Aplicación	6. Presentación
(RPC)		5. Sesión
(TCP/UDP)	4. Transporte	4. Transporte
(IP/ICMP)	3. Internet	3. Red
Trama Ethernet	2. Interfaz de Red	2. Enlace de Datos
Red Ethernet	1.Hardware	1. Físico

Trama Telnet

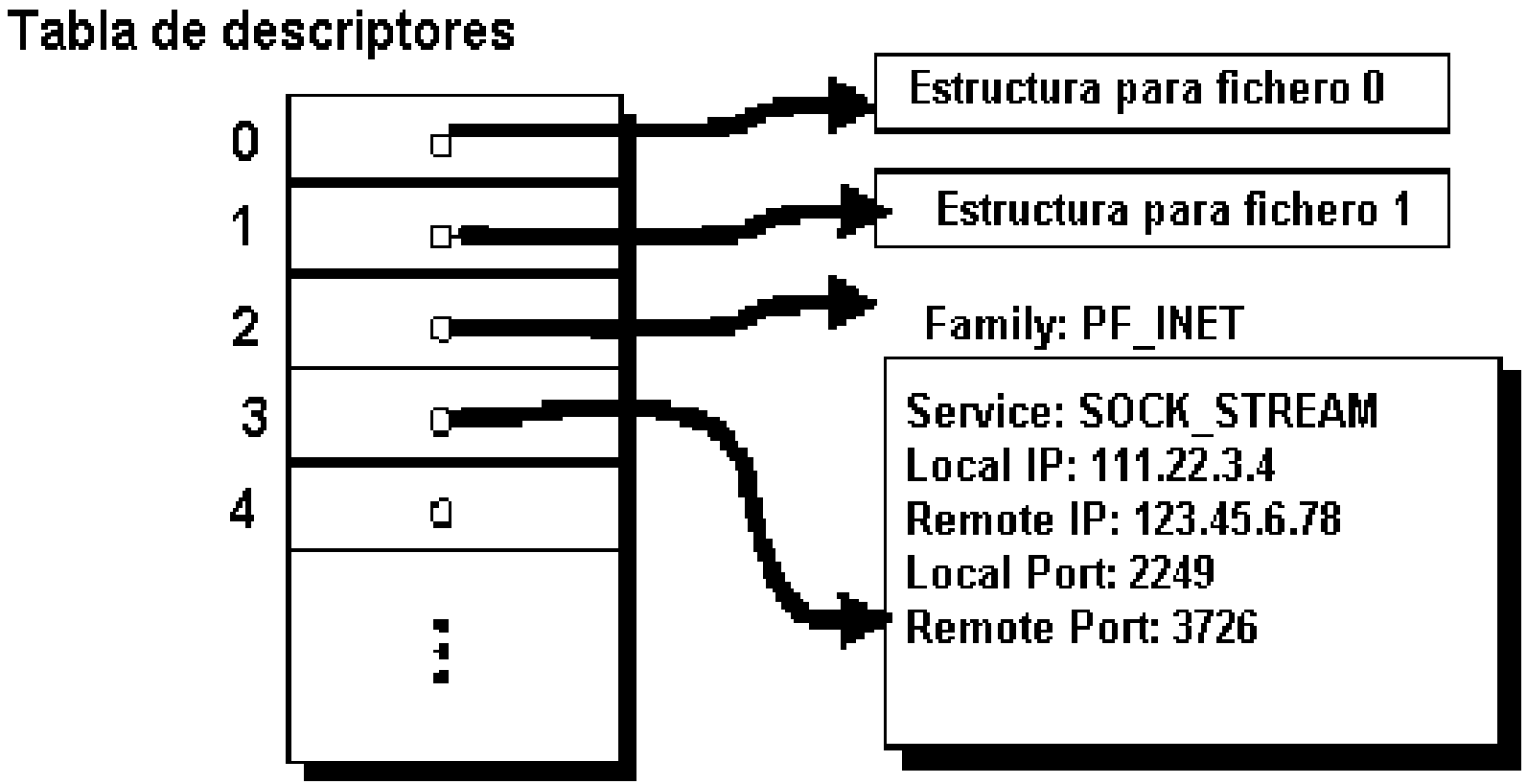


Esquema de comunicaciónm sockets





Entrada de descriptor de socket

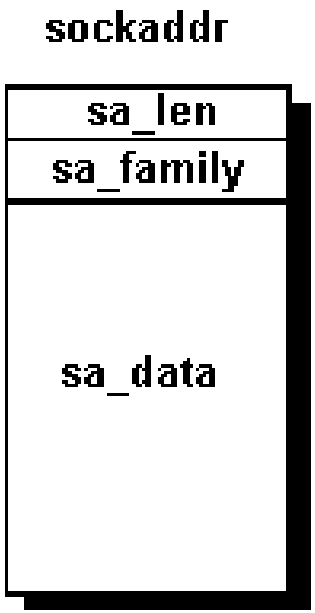


Estructura de Sockets

```
struct sockaddr
(
        u_short sin_family;
        char    sin_zro(14);
)
```

```
#include <netlinet/in.h>

struct sockaddr_in
{
        u_short sin_family;
        u_short sin_port;
        u_long  sin_addr;
        char    sin_zero();
}
```



System calls

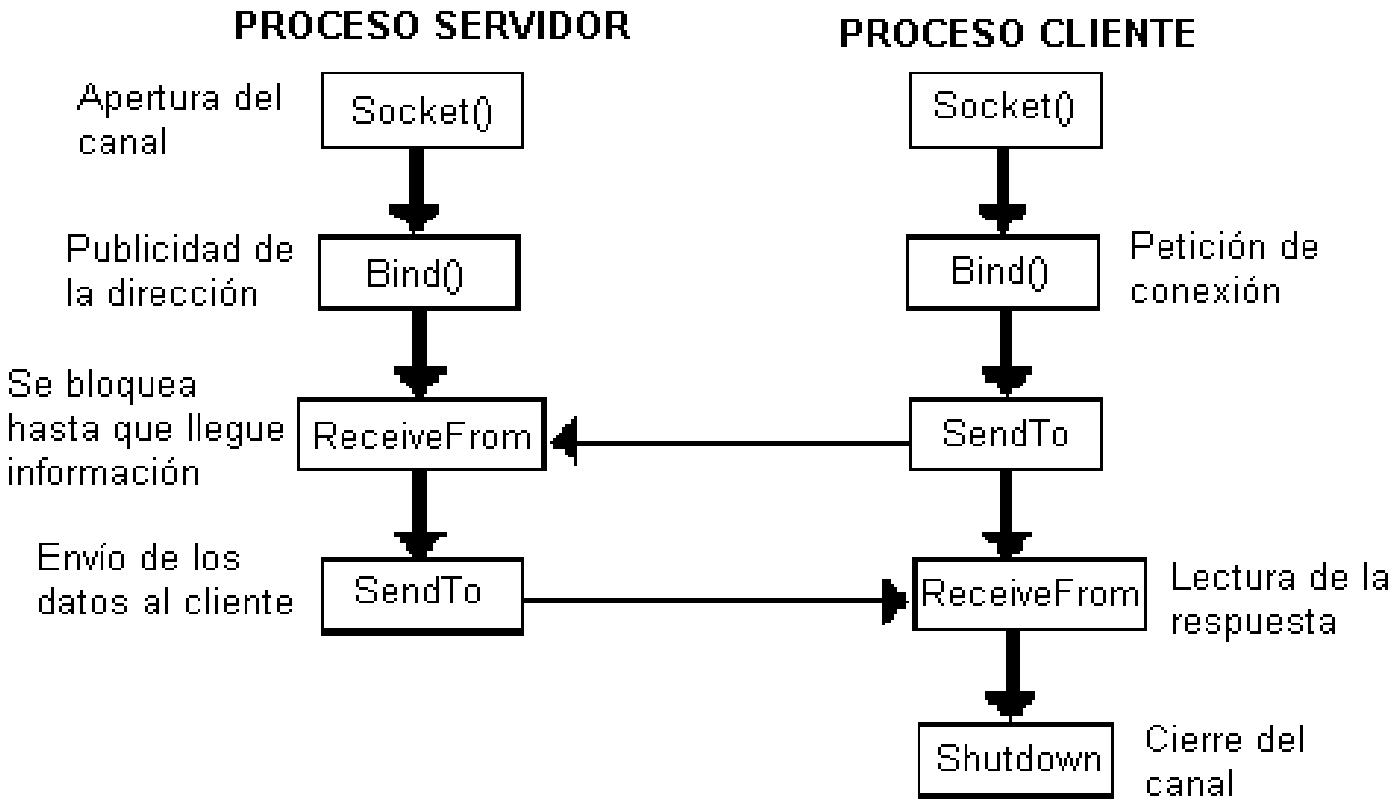
- socket (dominio, tipo, protocolo)
- Dominio = interno – externo
- Tipo = d-gram – stream – RAW
- Protocolo = UDP – TCP – ICMP

- Devuelve un sockdf

System calls

- socket (dominio, tipo, protocolo)
- bind (asocia socket con fd o puerto)
- listen (long. de pedido y queda a la escucha)
- accept (escucha al socket y acepta conexión
- hostent (obtiene información de nodo remoto)

Sin Conexón

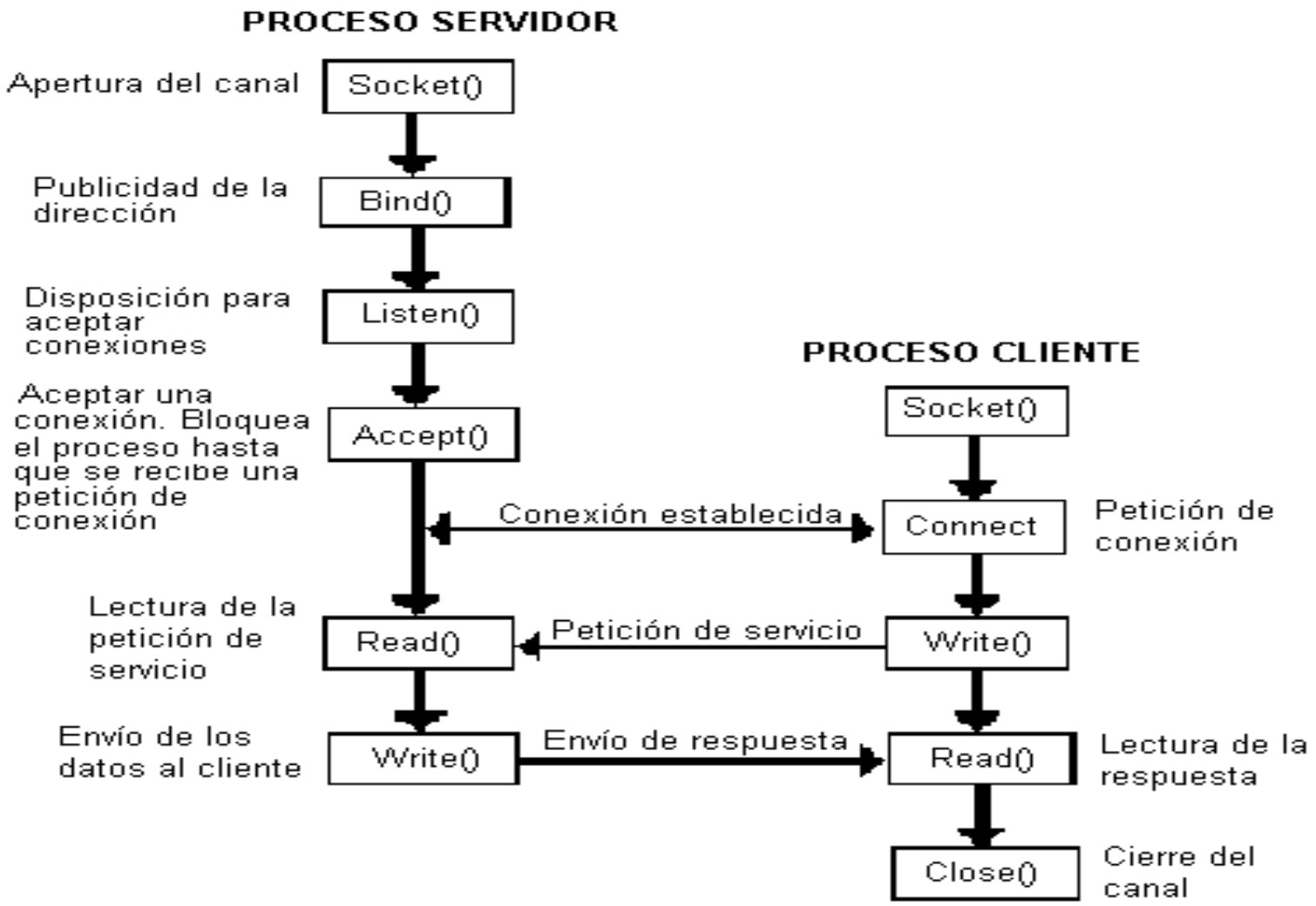

PROCESO SERVIDOR
PROCESO CLIENTE
Apertura del canal
Socket()
Socket()
Publicidad de la dirección
Bind()
Bind()
Petición de conexión
Se bloquea hasta que llegue información
ReceiveFrom
SendTo
Envío de los datos al cliente
SendTo
ReceiveFrom
Lectura de la respuesta
Shutdown
Cierre del canal

Sin Conexión

s = socket(AF_INET, SOCK_DGRAM, 0);
bind (s, sender_adress, server_address-length);
sendto (s, “msj”, receiver_address);
Close(s)

Receiver
s = socket(AF_INET, SOCK_DGRAM, 0);
bind (s, receiver_address, receiver_address-length);
amount = recvfrom (s, buffer, sender_address);
close (s)

Con Conexión



Con Conexión

Client

4) socket = (AF_INET, SOCK_STRAM, 0);

6) connect (s, server_addres, server_address_length);

7) write (s, “msj”, msg_length);

close (s)

Server

1) socket = (AF_INET, SOCK_STRAM, 0);

2) bind (s, server_address, server_address_length);

3) listen (s, backlog);

5) snew = accpet (s, client_address, client_address_length);

8) nbytes = read (snew, buffer, amount);

close (snew)

close (s)

Intercambio de Mensajes (Rendez Vous)

- send y receive bloqueantes

```
tarea Productor;
        begin
        <Producir un msg>;
        Send msg to Consumidor
        end;
tarea Consumidor;
        begin
        Receive msg from Productor;
        <Consumir msg>
        end;
```

Intercambio de Mensajes (Rendez Vous)

- send y receive bloqueantes
- **Abstracción**

V(x)
P(y)
SEND

P(x)
V(y)
RECEIVE

Intercambio de Mensajes (Rendez Vous extendido)

send y receive bloqueantes

Abstracción (necesita conocer un ack)

V(x)

SEND

P(y)

P(x)

RECEIVE

If ok Then V(y)

Intercambio de Mensajes (Rendez Vous asimétrico)

```
Tarea T1;
   begin
   Send x to T2;
   end;
Tarea T2;
   begin
   Accept Send (x);
   y := x;
   end;
```

Intercambio de Mensajes (Rendez Vous asimétrica y semisincrónica)

- **Asimétrica**: un send asincrónico continua su ejecución sin bloquearse.

  En el caso de receive asincrónico el receptor continúa su ejecución aunque no haya llegado nada

- **Semisincrónica**: **send** no bloqueantes y **receive** bloqueantes

Comunicación indirecta
Mailbox

- Indirecto: La comunicación se mantiene a través de una estructura de datos compartida (llamada mailbox)
- Ventaja: desacopla al emisor y al receptor, permitiendo mayor flexibilidad en el uso de los mensajes.
- Desventaja: Centraliza.

- La utilización del mailbox es posible en las modalidades:
  - Uno a uno, también llamado enlace privado.
  - Muchos a uno, que corresponde al modelo cliente/servidor.
  - Uno a muchos, también llamado broadcast.
  - Muchos a muchos, que se puede visualizar como la posibilidad de tener muchos servidores

- Por lo general los mailbox manejan disciplinas de colas en modalidad FIFO, pero también es posible el manejo por prioridades.

Comunicación indirecta
Mailbox

- Es posible manejar los modelos de:
- Exclusión
- Productor – Consumidor
- Lectores - Escritores

Mailbox excluyente

Se utilizan **receive bloqueantes y send no bloqueantes**.
El **mailbox** se lo utiliza como contenedor de un **token.**

```
/* programa exclusion-mutua */
int  n=   /* número de procesos */

void p(int i)
{
      mensaje msj;
      while (cierto)
      {
            receive (exmut, msj);  /*si el mailbox está vacío el proceso se detiene */
            /* sección crítica */
            send (exmut, msj);
            /* resto */
      }
}

void main ()
{
      crear-mailbox (exmut);
      send (exmut, token);
      parbegin (p1, p2, p3, ..., pn);
}
```

Mailbox Productor - Consumidor

Se utilizan **receive bloqueantes y send no bloqueantes**.
Se utilizan dos buzones, puede_consumir y puede_producir
Capacidad = /* capacidad del buffer */;
Int i;

```
Void productor()
{
    mensaje msjp;
    while (cierto);
    {
        receive (puede_producir, msjp);
        msjp = producir();
        send (puede-consumir, msjp);
    }
}
```

Mailbox Productor - Consumidor

```
void consumidor()
{
    mensaje msjc;
    while (cierto)
    {
        receive (puede_consumir, msjc);
        consumir(msjc);
        send (puede_producir, token);
    }
}
```

Mailbox Productor - Consumidor

```
void main()
{
    crear_mailbox (puede_producir);
    crear_mailbox (puede_consumir);
    for (int i = 1, i <= capacidad; i++)
            send (puede_producir, token);
            send (puede_consumir, null);
    parbegin (productor, consumidor) parend;
}
```

Mailbox Lectores - Escritores

- Se utilizan **3 mailbox pedir_lectura pedir-escritura** y **terminado.**
- **Además de los procesos lector y escritor se utiliza uno auxiliar llamado controlador** que actúa según el valor de una variable **cont, según lo siguiente:**

- **Cont > 0 no hay escritores esperando**
- **Cont = 0 pendientes escrituras, esperar terminado**
- **Cont < 0 escritor en espera**

- **Cantidad máxima de lectores = 100**

Mailbox Lectores - Escritores

```
void lector(int i)
{
    mensaje msjl;
    while (cierto)
    {
        msjl = i;
        send (pedir-lectura, msjl);
        receive (buzón[i], msjl);
        LEER;
        msjl = i;
        send (terminado, msjl);
    }
}
```

Mailbox Lectores - Escritores

```
void escritor(int j)
{
    mensaje msje;
    while (cierto)
    {
        msje=j;
        send (pedir_escritura, msje);
        receive (buzón[j], msje);
        ESCRIBIR;
        msje = j;
        send (terminado, msje)
    }
}
```

Mailbox Lectores - Escritores

```
void controlador()
{
    while(cierto)
    {
        if cont > 0
        {
            if (!vacío (terminado))
            {
                receive (terminado, msj);
                cont++
            }
            else
            if (!vacío (pedir_escritura))
```

Mailbox Lectores - Escritores

```
{
                    receive (pedir_escritura, msj);
                    escritor_id = msj-id;
                    cont  = cont – 100;
                }
                else
                if (!vacío (pedir_lectura))
                    {
                    receive (pedir_lectura, msj));
                    cont--;
                    send(msj_id, “OK”);
                }
            }
```

Mailbox Lectores - Escritores

```
if (cont == 0)
        {
                send (escritor_id, "OK");
                receive (terminado, msj);
                cont = 100;
        }
        while (cont < 0)
        {
                receive (terminado, msj);
                cont ++;
        }
    }
}
```