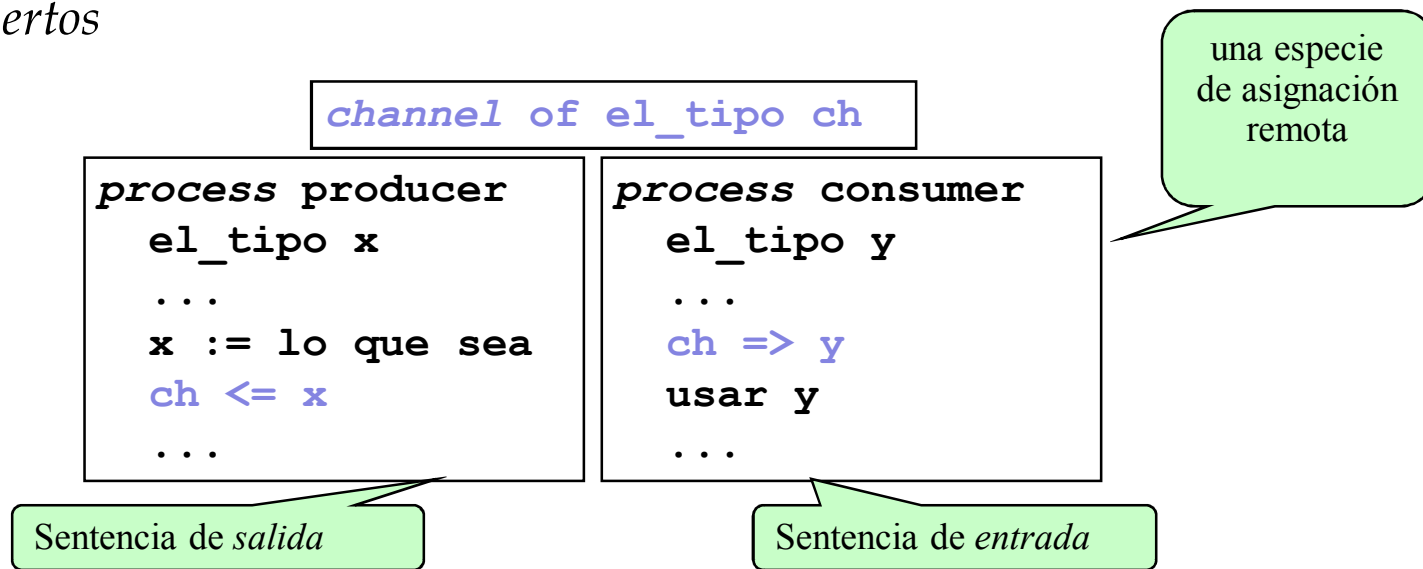


Lección 9: Programación mediante paso síncrono de mensajes

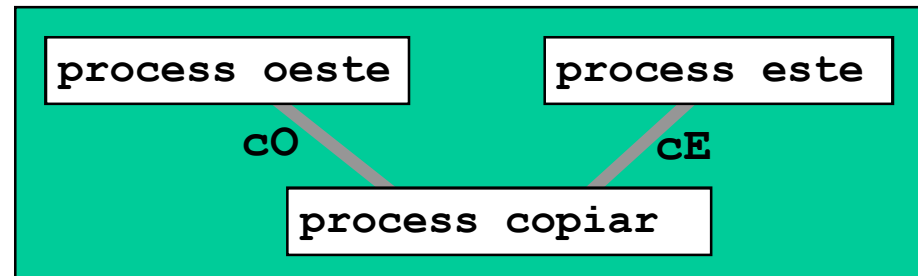
- Paso síncrono de mensajes:
 - notación simplificada
 - ejemplo: proceso filtro “copiar”
 - notación general
 - ejemplo: proceso servidor “mcd”
- Entrada selectiva
 - o cómo esperar en varios canales a la vez
- Ejercicios
- Citas y RPC

Paso síncrono de mensajes

- Concepto y notación usados por Hoare (1978)
 - Communicating Sequential Processes (CSP)
- La comunicación entre procesos se va a llevar a cabo mediante *puertos*



Paso síncrono de mensajes



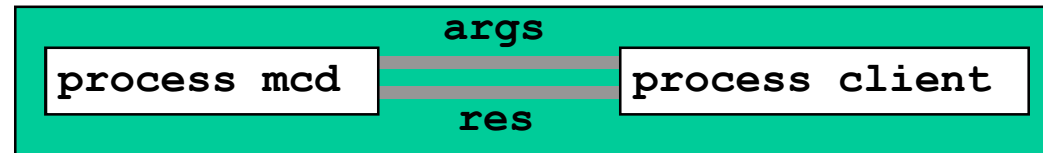
channel of character cE, cO

```
process oeste
character c
loop
  cO => c
  ...
```

```
process copiar
character c
loop
  cE => c
  cO <= c
```

```
process este
character c
loop
  produce nuevo c
  cE <= c
```

Paso síncrono de mensajes

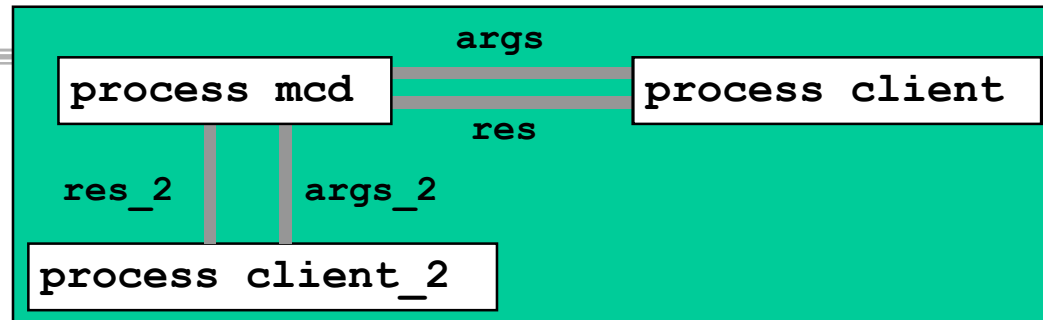


```
channel of (integer,integer) args
channel of integer res
```

```
process mcd
  integer x,y,m
  integer r := 1
  loop
    args => (x,y)
    ... --calcula mcd en m
    res <= m
```

```
process client
  integer a,b,m
  loop
    obtener a,b
    args <= a,b
    res => m
```

Entrada selectiva



- ¿Qué pasa cuando hay más de un cliente?
 - Solución 1: directa
 - Solución 2: escuchar selectivamente en varios canales

```
either ch1 => var1
      ...
or     ch2 => var2
      ...
or     ch3 => var3
      ...
```

```
either i:1..n
      ch[i] => var
      ...
```

Entrada selectiva

```
channel of (integer, integer) args, args_2
channel of integer res, res_2
```

```
process mcd
  integer x, y, m, cl
  integer r := 1
  loop
    either args => (x, y)
      cl := 1
    or   args_2 => (x, y)
      cl := 2
    ... --calcula mcd en m
  if cl=1
    res <= m
  else --cl=2
    res_2 <= m
```

```
process client
  integer a, b, m
  loop
    obtener a, b
    args <= a, b
    res => m
```

```
process client_2
  integer a, b, m
  loop
    obtener a, b
    args_2 <= a, b
    res_2 => m
```

Entrada selectiva

```
channel of (integer, integer) args, args_2
channel of integer res, res_2
```

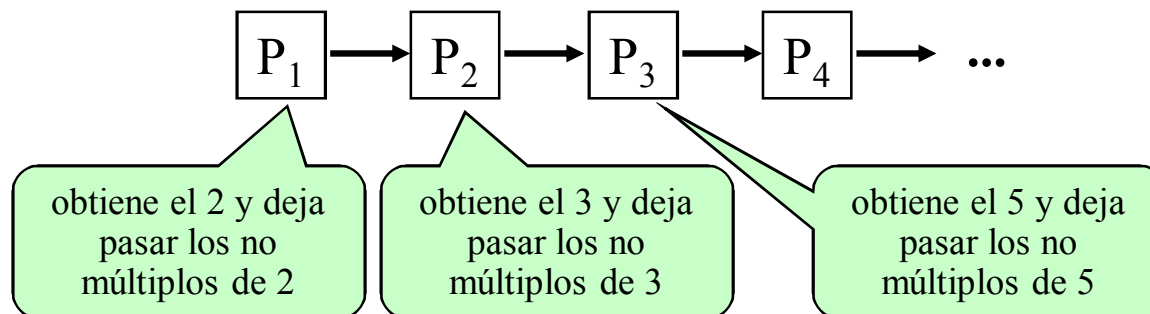
```
process mcd
  integer x, y, m, cl
  integer r := 1
  loop
    either args => (x, y)
      ... --calcula mcd en m
      res <= m
    or
      args_2 => (x, y)
      ... --calcula mcd en m
      res_2 <= m
```

```
process client
  integer a, b, m
  loop
    obtener a, b
    args <= a, b
    res => m
```

```
process client_2
  integer a, b, m
  loop
    obtener a, b
    args_2 <= a, b
    res_2 => m
```

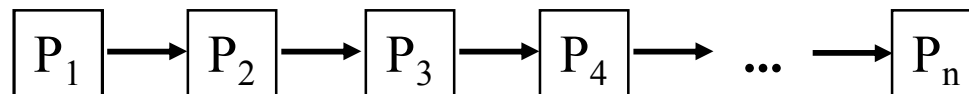
Ejemplo: la criba de Eratóstenes

- **La criba de Eratóstenes:** encuentra los primos menores o iguales que un n dado
- Secuencial sencillo:
 - el 2 es primo; “tachar” su múltiplos.
 - el primero no “tachado” (el 3) es primo; tachar sus múltiplos
 - el primero no “tachado” (el 5) es primo; tachar sus múltiplos. etc.
- Una versión en “pipe-line”:



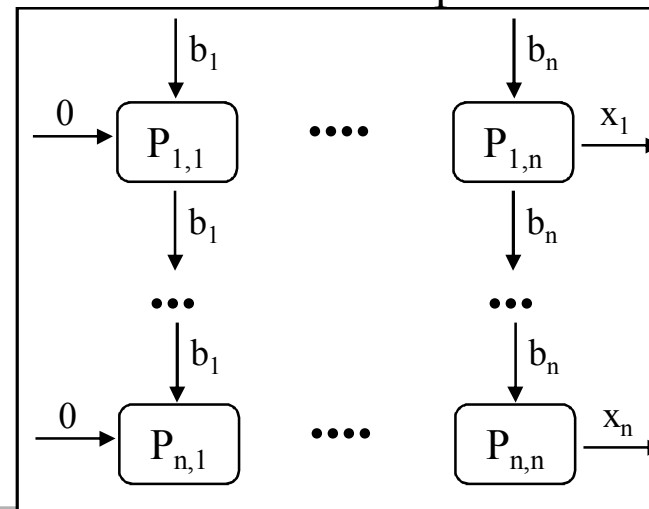
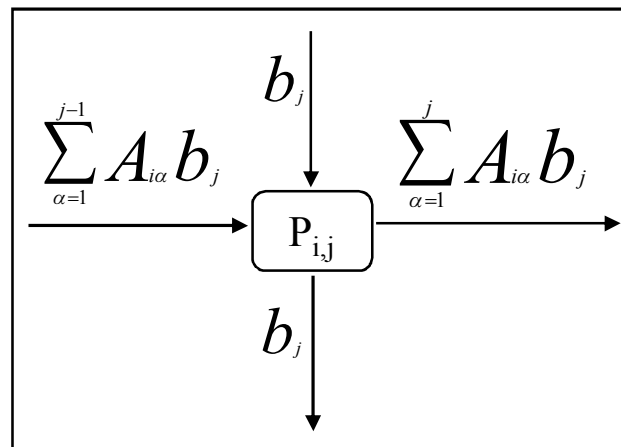
Ejemplo: "Pipeline sort"

- **"Pipeline sort"**: utilizando n procesos y con un flujo de datos como el que se esquematiza en la figura, proponer un algoritmo de ordenación para n datos, de manera que al terminar cada proceso tiene el valor que le corresponde de la secuencia de datos, según el orden ascendente



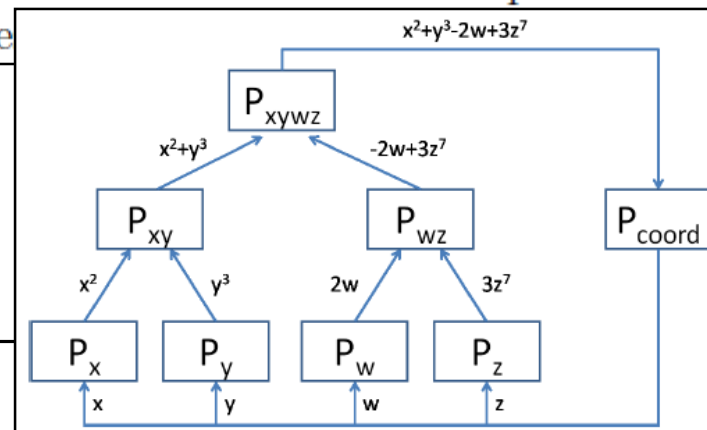
Ejemplo: producto matriz por vector

- **Problema:** calcular " $\mathbf{x}=\mathbf{A}\mathbf{b}$ " siendo " \mathbf{A} " una matriz $n \times n$ y " \mathbf{x} " y " \mathbf{b} " vectores $n \times 1$
- Una solución síncrona, usando $n \times n$ procesadores:
 - V1: cada $P(i,j)$ tiene almacenado $A[i,j]$
 - V2: el valor $A[i,j]$ se tiene que suministrar en una fase previa



Ejemplo: Árbol de evaluación

La figura 1 muestra la organización de un sistema en el que mediante siete procesos $P_x, P_y, P_w, P_z, P_{xy}, P_{wz}, P_{xywz}$, más un proceso coordinador P_{coord} , se evalúa la expresión $x^2 + y^3 - 2w + 3z^7$, donde los parámetros x, y, w y z son números reales. Para ello, en un bucle infinito, el proceso coordinador lee de la entrada estándar los valores de los parámetros, los suministra a los procesos P_x, P_y, P_w, P_z y espera el resultado que le suministrará el proceso P_{xywz} para mostrarlo por la salida estándar. Cada proceso realiza un cálculo parcial, que suministra a otro proceso.



El ejercicio pide:

- Definir la red de canales síncronos para resolver el problema
- Escribir del código de los ocho procesos, de acuerdo a la especificación dada

Ejercicio: los filósofos

```
channel of integer[1..n] canFil
channel of (integer,integer) entrada
```

```
process filosofo(i:1..n)
  integer kk

  loop forever
    --piensa
    entrada <= (i,COGER)
    canFil[i] => kk
    --come
    entrada <= (i,DEJAR)
```

```
process servidor
  integer who,what
  set of integer tenLibres := {1..n}
  set of integer fileEsperan := {}
  loop forever
    entrada => (who, what)
    switch
      what=DEJAR: ...
      what=COGER: ...
    end switch
  end loop
```

Ejercicio: los filósofos

```
switch      --operaciones MOD n
  what=COGER:
    if who IN libres AND who+1 IN libres
      canFil[who] <= 1
      libres := libres \ {who,who+1}
    else
      esperan := esperan UNION {who}
    end if
```

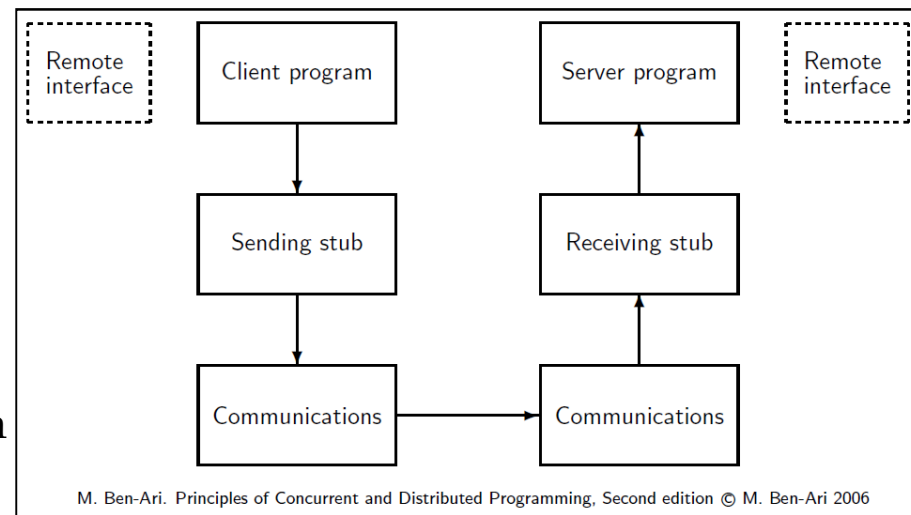
Ejercicio: los filósofos

```
switch      --operaciones MOD n
  what=COGER: . . .
  what=DEJAR:
    if (who-1 IN libres) AND (who-1 IN fileEsperan)
      canFil[who-1] <= 1
      libres := libres \ {who-1}
      fileEsperan := fileEsperan \ {who-1}
    else
      libres := libres UNION {who}
    end if

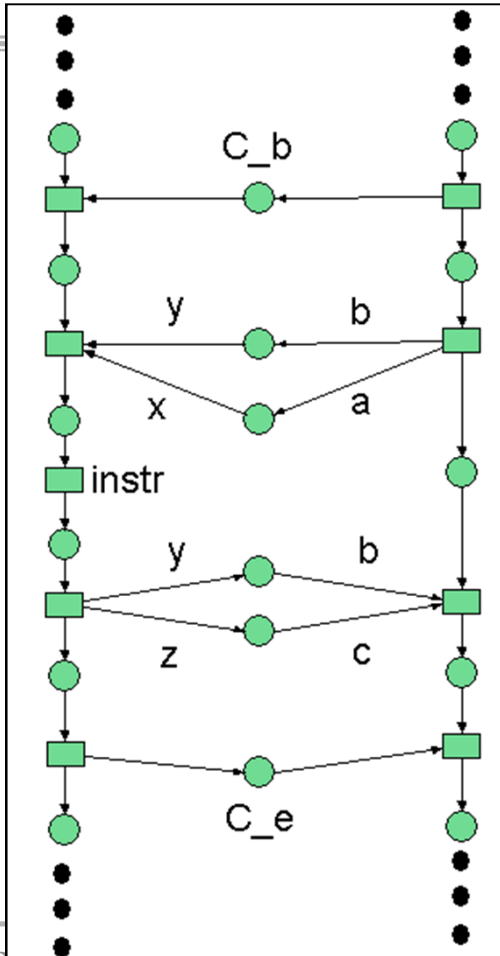
    if (who+1 IN fileEsperan) AND (who+2 IN libres)
      canFil[who+1] <= 1
      libres := libres \ {who+2}
      fileEsperan := fileEsperan \ {who+1}
    else
      libres := libres UNION {who+1}
    end if
```

RPC

- **Remote Procedure Call:** un cliente invoca servicios de un servidor que puede estar en otro procesador
 - cliente invoca como una operación normal
 - el servidor pone en marcha un proceso para atenderlo
- Servidor y cliente se deben compilar con un interfaz común
 - para parámetros y operaciones
- **RMI** para Java
 - Remote Method Invocation



Citas



```

entry C(x: in ...;
        y: in out ...;
        z: out ...);
    ...
task body T1 is
begin
    ...
    accept C(x,y,z) do
        instr
    end C2;
    ...
end T1;
    
```

asimetría

```

task body T2 is
begin
    ...
    T1.C(a,b,c);
    ...
end T2;
    
```


Canales síncronos en C++ y Java

`channel of T canal`

process P1

T d1

...

canal => d1

...

canal <= exp

...

process P2

T d2

...

canal => d2

...

canal <= exp2

...

process P3

T d3

...

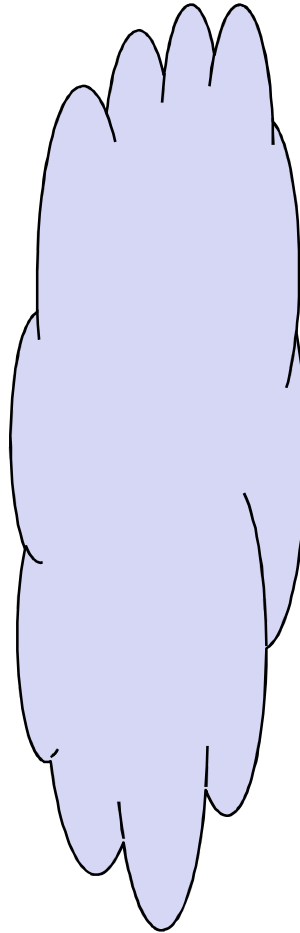
canal => d3

...

canal <= exp3

...

channel of T canal



P1

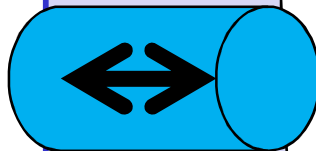
P2

```
Socket cS (SERVER_PORT);
```

```
canal == port (no tipado)
```

```
int socket_fd = cS.Bind();
```

```
int error_code = cS.Listen(nC);
```



P1

SERVIDOR (IP)

155.210.***.***

P2

CLIENTE

J. Ezpeleta-P. Álvarez
Univ. de Zaragoza

19

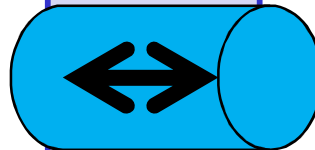
```
Socket cS(SERVER_PORT);
```

```
int socket_fd = cS.Bind();
```

```
int error_code = cS.Listen(nC);
```

```
canal == port (no tipado)
```

```
Socket cC(SERVER_ADDRESS,  
SERVER_PORT);
```



P1

SERVIDOR (IP)

155.210.***.***

P2

CLIENTE

J. Ezpeleta-P. Álvarez
Univ. de Zaragoza

20

```
Socket cS (SERVER_PORT);
```

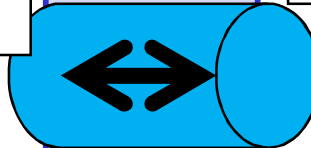
```
int socket_fd = cS.Bind();
```

```
int error_code = cS.Listen(nC);
```

```
int client_fd = cS.Accept();
```

```
Socket cC (SERVER_ADDRESS,  
SERVER_PORT);
```

```
int socket_fd = cC.Connect();
```



P1

SERVIDOR (IP)

155.210.***.***

P2

CLIENTE

J. Ezpeleta-P. Álvarez
Univ. de Zaragoza

21

```
Socket cS(SERVER_PORT);
```

```
int socket_fd = cS.Bind();
```

```
int error_code = cS.Listen(nC);
```

```
int client_fd = cS.Accept();
```

```
int rcv_bytes = cS.Rcv(  
    client_fd, buffer, length);
```

```
Socket cC(SERVER_ADDRESS,  
    SERVER_PORT);
```

```
int socket_fd = cC.Connect();
```

```
int send_bytes = cC.Send(  
    socket_fd, message);
```

P1

SERVIDOR (IP)

155.210.***.***

P2

CLIENTE

J. Ezpeleta-P. Álvarez
Univ. de Zaragoza

22

```
Socket cS(SERVER_PORT);
```

```
int socket_fd = cS.Bind();
```

```
int error_code = cS.Listen(nC);
```

```
int client_fd = cS.Accept();
```

```
int rcv_bytes = cS.Rcv(  
    client_fd, buffer, length);
```

```
int send_bytes = cS.Send(  
    client_fd, buffer);
```

```
Socket cC(SERVER_ADDRESS,  
    SERVER_PORT);
```

```
int socket_fd = cC.Connect();
```

```
int send_bytes = cC.Send(  
    socket_fd, message);
```

```
int read_bytes = cC.Rcv(  
    socket_fd, buffer, length);
```

P1

SERVIDOR (IP)

155.210.***.***

P2

CLIENTE

J. Ezpeleta-P. Álvarez
Univ. de Zaragoza

23

```
Socket cS(SERVER_PORT);
```

```
int socket_fd = cS.Bind();
```

```
int error_code = cS.Listen(nC);
```

```
int client_fd = cS.Accept();
```

```
int rcv_bytes = cS.Rcv(  
    client_fd, buffer, length);
```

```
int send_bytes = cS.Send(  
    client_fd, buffer);
```

```
error_code = cS.Close(  
    client_fd);
```

```
error_code = cS.Close(  
    socket_fd);
```

```
Socket cC(SERVER_ADDRESS,  
    SERVER_PORT);
```

```
int socket_fd = cC.Connect();
```

```
int send_bytes = cC.Send(  
    socket_fd, message);
```

```
int read_bytes = cC.Rcv(  
    socket_fd, buffer, length);
```

```
int error_code = cC.Close(  
    socket_fd);
```

P1

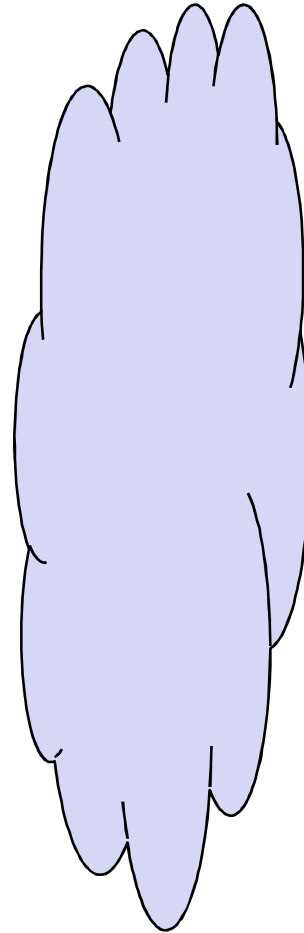
SERVIDOR (IP)

155.210.***.***

P2

CLIENTE

channel of T canal



P1

P2

```
ss=new ServerSocket(PORT#)
```

```
can=ss.accept()
```

canal ==== port (no tipado)

```
import java.net.Socket;  
import java.net.ServerSocket;
```

P1
SERVIDOR (IP)
155.210.***.***

P2
CLIENTE

