IPC

Inter-process Communication

¿Qué es?

- Función básica de un OS.
- Permite enviar información (datos o estado) entre procesos
- Diferentes técnicas

Necesidades

- Compartir información (db)
- Acelerar los cálculos (multicore)
- Modularidad (1 funcion por proceso)
- Conveniencia (muchas tareas simultaneamente)

Clasificación de distintas IPC

- Paso de Mensajes

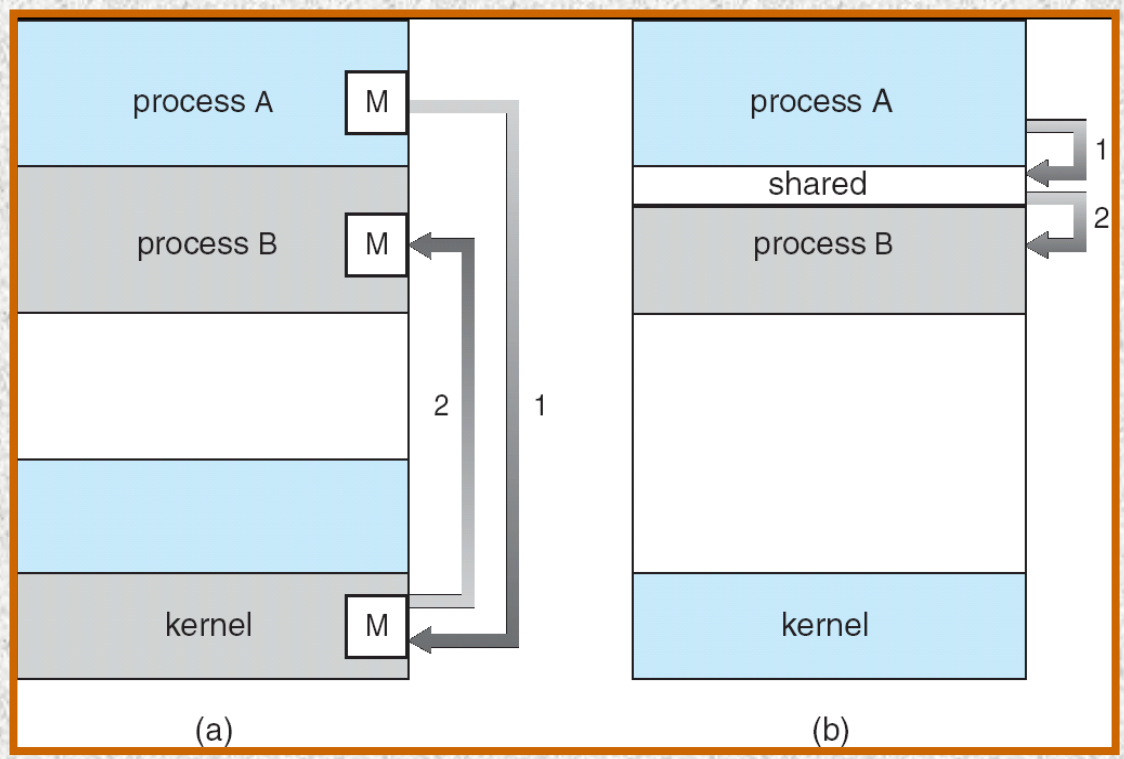
  Usando llamados al kernel

  Pequeñas cantidad de información

  Fácil de implementar

Memoria Compartida

- Comparte espacio de mem.
- Rápida



Implementaciones de IPC

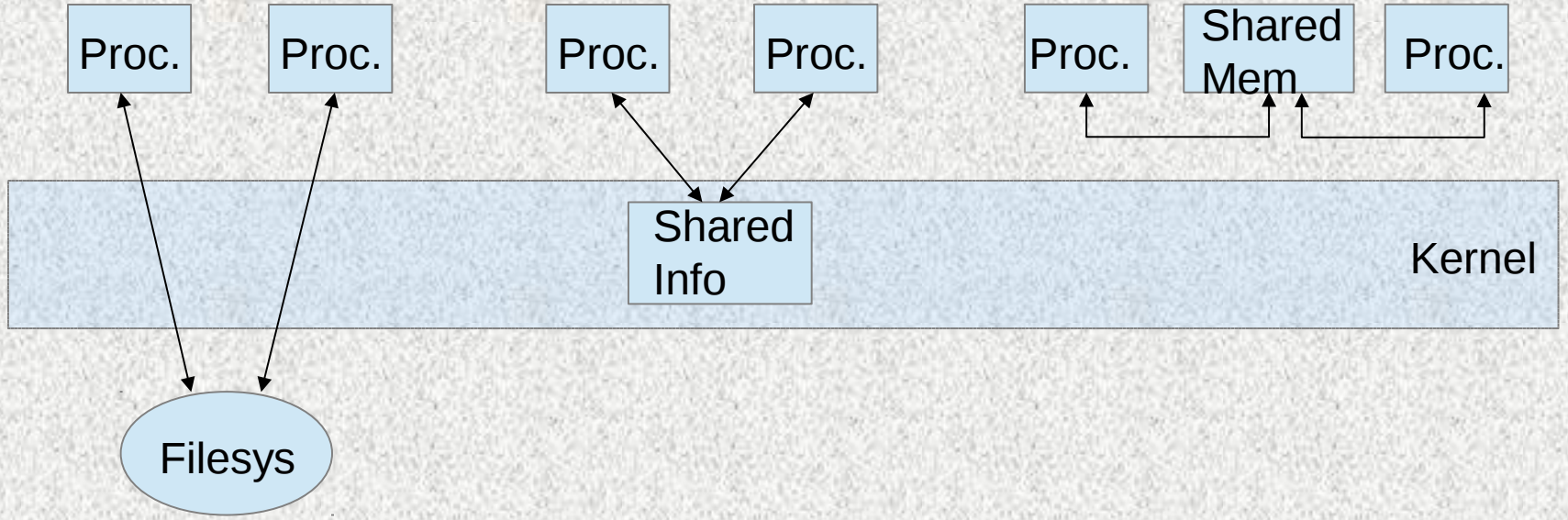
- Paso de Mensajes:
  - Pipe (Tuberías anónimas)
  - FIFO (Tuberías con nombre)
  - Cola de mensajes
  - MPI (interfaz de paso de mensajes)
  - RPC (llamadas de procedimiento remotas)
  - Sockets
- Memoria compartida
  - Mapeo de archivos anónimos
  - Objetos de memoria
  - Archivos de memoria

IPC y los caminos para compartir información

Sistema de archivos: Archivo o Socket

Kernel: Pipe, FIFO, Cola de mensajes

Memoria: Memoria compartida



Persistencia

Cuanto tiempo permanece en existencia

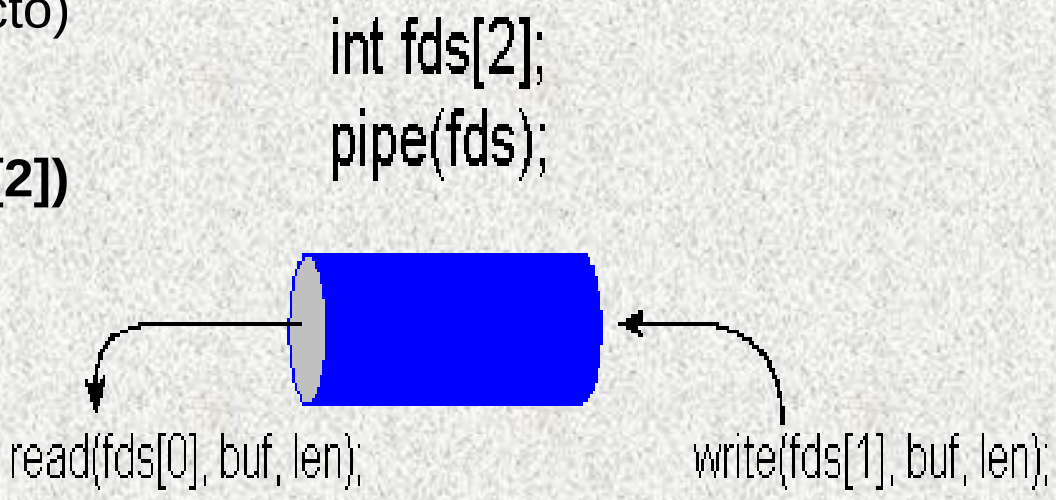
- Proceso:
  - Pipe
  - Fifo (cuidado no es persistencia de kernel, aunque persiste cuando todos los procesos involucrados terminaron, no los datos de los mismos!)
  - Socket
  - RPC
  - MPI
  - Memoria compartida (mapeo de archivos anónimos)
- Kernel:
  - Cola de mensajes
  - Memoria compartida (objetos de memoria)
- Filesystem
  - Memoria compartida (archivos de memoria)

Uso

- Comunicación entre procesos de un mismo PC (los que analizaremos):
  - Pipes
  - Fifo
  - Colas de mensajes
  - Memoria compartida
- Comunicación entre procesos de una misma PC o de distintas PCs en una red :
  - Sockets
  - RPC
  - MPI

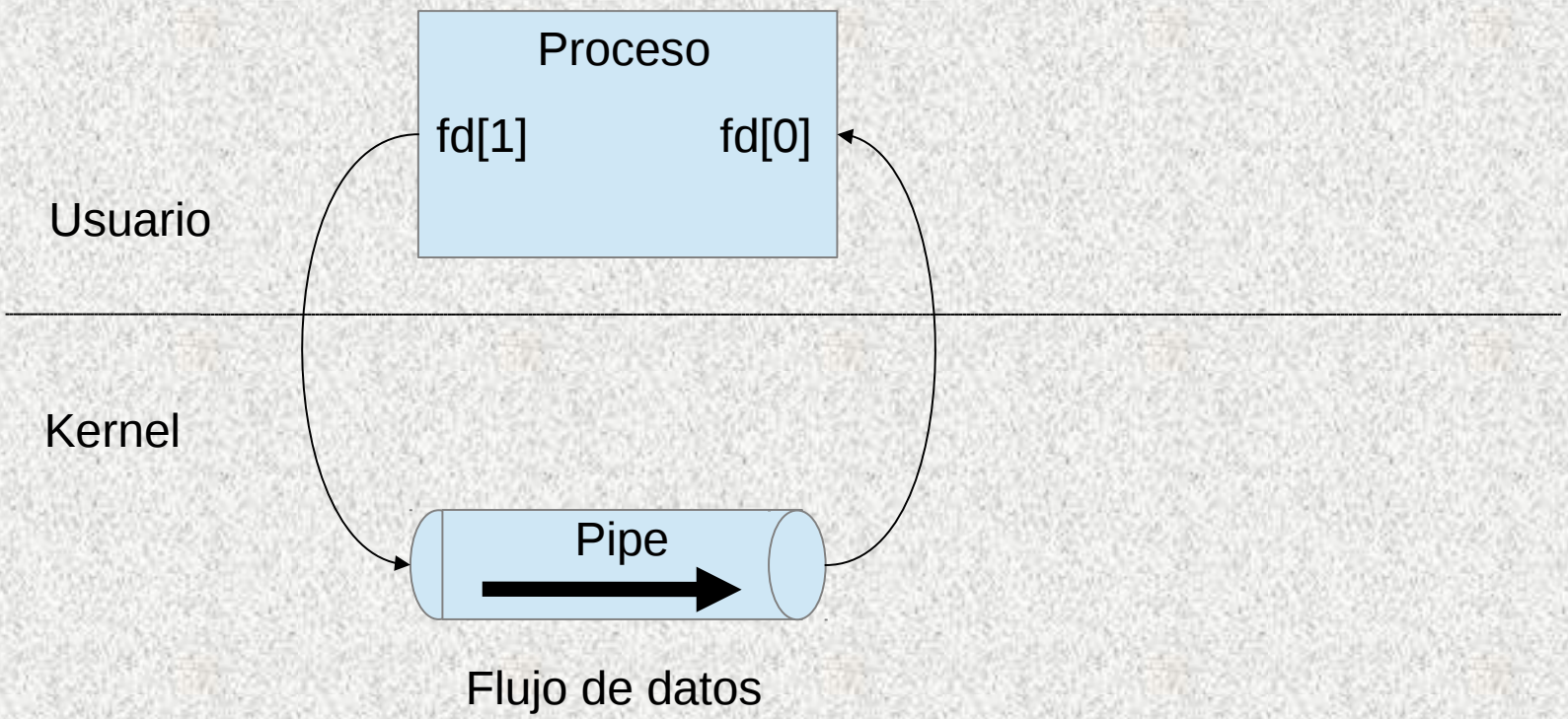
Pipe – Tuberías anónimas

- Permiten comunicar información entre procesos RELACIONADOS por herencia
- Mensajes orientados a “stream” de datos (no segmentados)
- No permiten compartir información entre procesos no relacionados
- Se pueden abrir configuradas como:
  - Bloqueantes (por defecto)
  - No bloqueantes
- Syscall **pipe(descritores[2])**



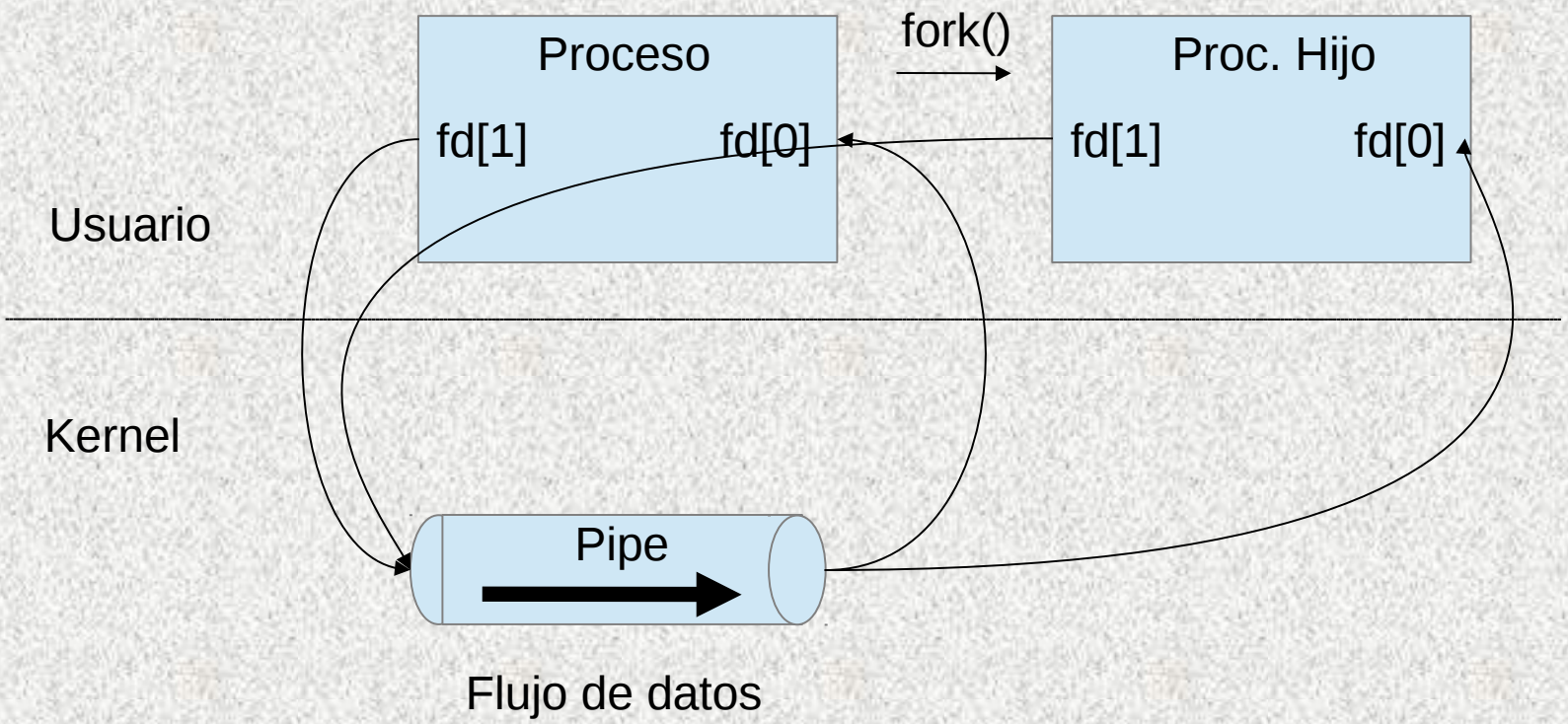
Pipe – Tuberías anónimas

- Ejemplo de uso



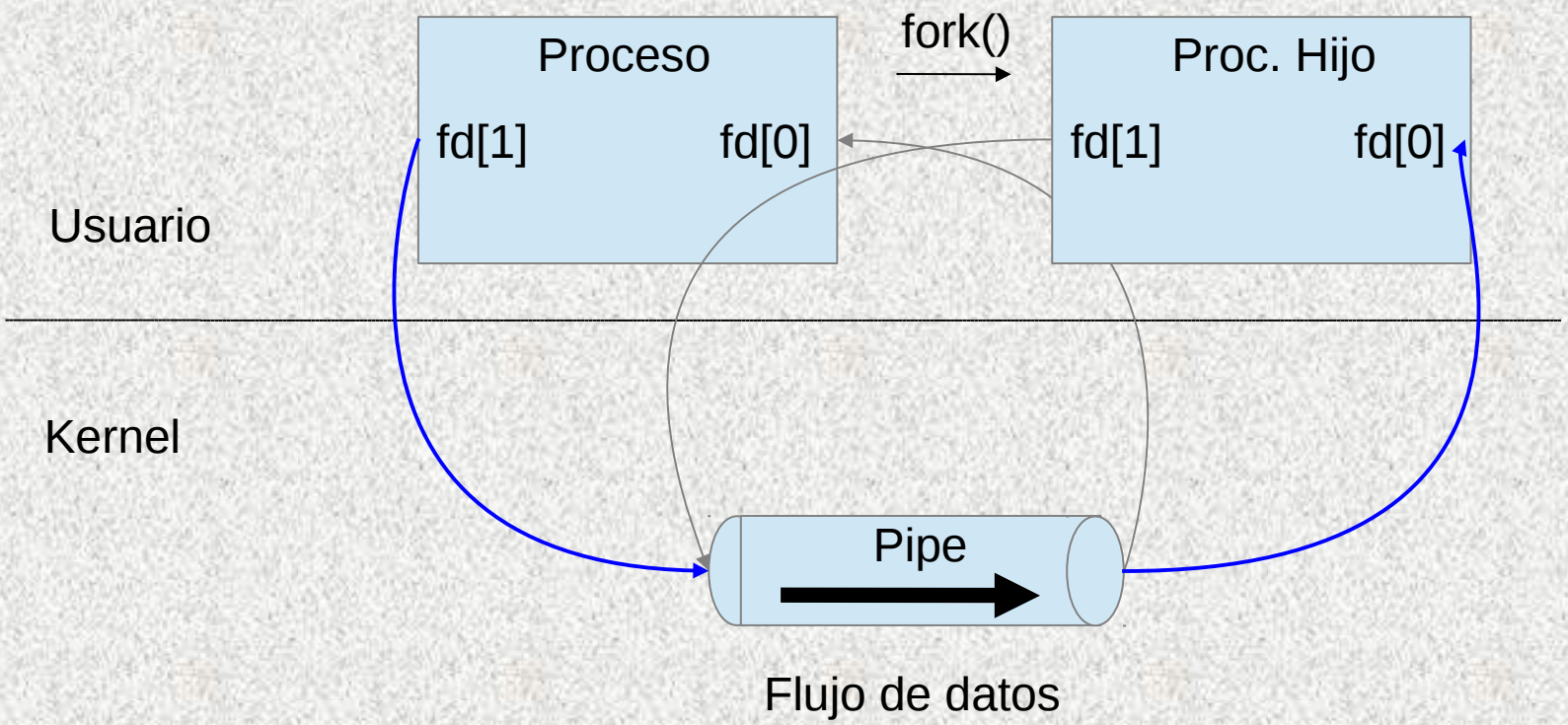
Pipe – Tuberías anónimas

- Ejemplo de uso Cont.



Pipe – Tuberías anónimas

- Ejemplo de uso Cont.



FIFO : Tuberías con nombre

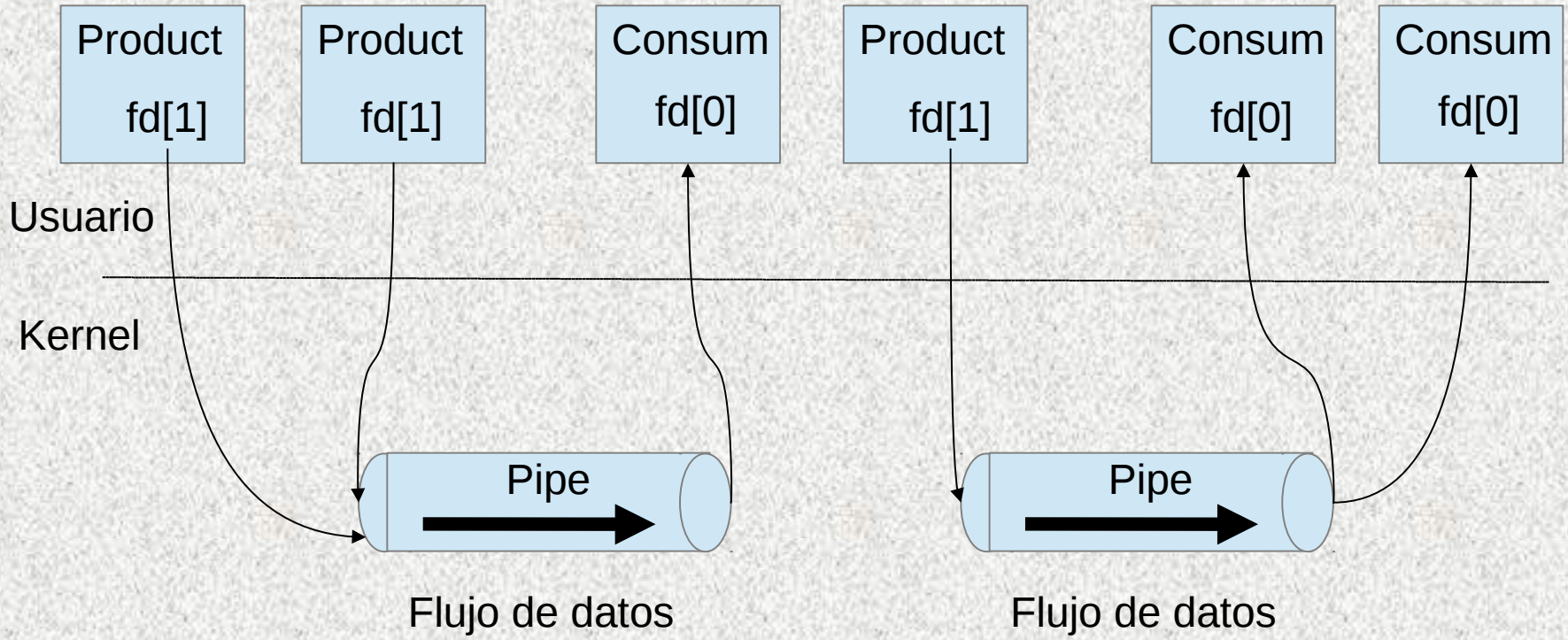
- Permite compartir información entre diferentes procesos no relacionados
- Mensajes orientados a stream de datos (no segmentados)
- La tubería persiste con un nombre en el filesystem
- Se pueden abrir configuradas como:
  - Bloqueantes
  - No bloqueantes
- Syscall **mkfifo(nombre,permisos)**

Ejemplo: mkfifo nombre , cat nombre , ls -l > nombre

ls -l > nombre , cat nombre

Problema de Stream de datos

- No discrimina que productor escribió los datos consumidos
- El primer consumidor lee los datos y el resto no



Problema de Stream de datos

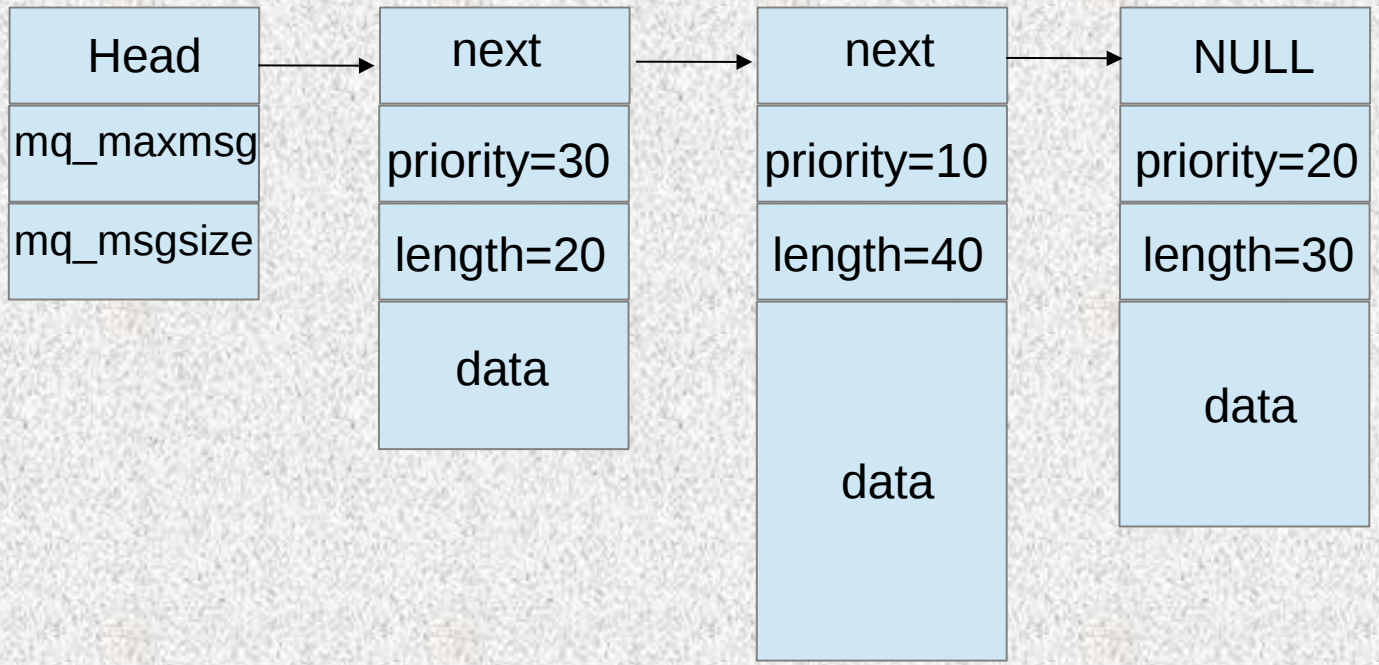
- Soluciones :
  - Caracteres especiales “in-band”
  - Largo explícito (se envía primero el size)
  - Largo fijo
  - Un registro por conexión (no aplicable a pipes .. solo a sockets)

Cola de mensajes (posix)

- Mensajes segmentados
- Permite compartir datos entre procesos sin que los mismos tengan “tiempo de convivencia” en común
- Permite establecer prioridades para reordenar mensajes en el receptor de acuerdo a su importancia
- Se pueden abrir configuradas como:
  - Bloqueantes
  - No bloqueantes
- descriptor=mq_open(nombre,banderas,permisos,…)
- mq_recieve/send(descriptor,info,longitud,prioridad)

Cola de mensajes

- Gralmente se implementa como lista enlazada



Cola de mensajes

- Syscall's usadas:
  - mqd_t **mq_open**(const char *name, int oflag);
  - int **mq_close**(mqd_t mqdes);
  - int **mq_send**(mqd_t mqdes, const char *msg_ptr, size_t msg_len, unsigned msg_prio);
  - ssize_t **mq_receive**(mqd_t mqdes, char *msg_ptr, size_t msg_len, unsigned *msg_prio);