Tuberías y FIFOs

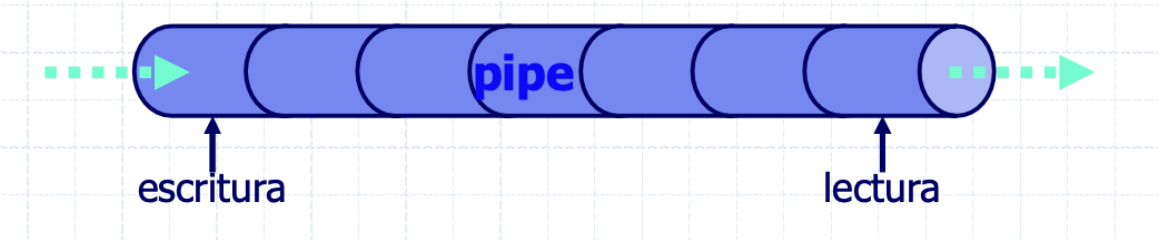
Tuberías (pipes)

- La forma más antigua de IPC en unix
  - Esta presente desde 1973; se encuentra, por tanto, en todas las variantes de Unix existentes
  - Los unix más antiguos no tienen memoria compartida debido a que el hardware de la PDP-11 hacía difícil su implementación
    - PDP
      - mini ordenador
  - No existe un UNIX que no tiene tuberías
- Una tubería permite la transmisión fiable de una cadena o chorro de bytes entre dos procesos
  - Comparable a una cola tipo FIFO
  - fiable llega lo que mando no se cambia
  - chorro significa que es continuado
    - no sabes si lo escribi en muchos chorros pero tu lo lees como 1 lectura
      - escribo 3 veces pero tu lo lees todo en 1 lectura
- Memoria compartida
  - Sistema de memoria virtual
    - traduce direcciones a memoria real
- Limitaciones de las tuberías
  - Son half-duplex (unidireccionales)
  - Sólo pueden ser usadas entre procesos que tienen un padre común
  - Presentan un tamaño máximo fijo para el buffer
    - buffer
      - zona de almacenamiento temporal
    - es del kernel
      - cachito de memoria
      - escribo con llamadas al sistema

Gestión de las tuberías

- La gestión de las tuberías está integrada en el sistema de ficheros
  - Implementadas clásicamente en el sistema de ficheros
  - Posteriormente como un caso particular, mediante sockets (4.3 BSD) o STREAMS (SVR4)
- Constituyen un canal de comunicación

    - Los datos datos escritos en un extremo del canal se leen en el otro extremo
    - La tubería usa un buffer que define su tamaño



7 bytes

- cuando escribo 7 la tubería esta llena
- debería leer el destino para liberar la tubería

- Dos cosas de unix
    - Proceso y ficheros
    - tuberías parecen ficheros
    - tipos de ficheros
        - ficheros y especiales
            - especiales
                - ratón
                - pantalla
                - modo bloque y modo caracter
                    - los de bloque se llaman asi porque tienen block buffer cache
                        - floppy
                        - disco
                        - guarda algunos sectores de los discos en la ram
                        - memoria ram es del kernel
                    - los de caracter no la tienen
                        - teclado
            - ficheros
                - todos son iguales
        - directorios
            - guardan nombres

Open Read Write Close

- open se comprueban los permisos
- Una vez abro el fichero puedo leer y escribir si me lo permite
- Es secuencial

Descriptor de ficheros

- Número
- Indice en la tabla de descriptor de ficheros

Que se escribe en un descriptor

- write(3, "hola", 14 bytes)
- lseek te permite mover el puntero

No hay forma de saber si escribimos de 3 golpes o 200 golpes el fichero no lo dice

- Hace que todos los ficheros sean iguales
- montones de bytes
    - con un tamaño

write le decimos al kernel escribe esto, el proceso se queda suspendido/dormido ya que se lo dice el kernel, el kernel se pone a trabajar, el kernel termina le dice ya termine, pasa a activo ya que esta terminado.

read es lo mismo

En las tuberías pasa lo mismo, si la tubería esta llena tiene que espera que la lean, entonces esta dormido el proceso

Si el lector lee la tubería y no hay nada se queda esperando hasta que alguien escriba

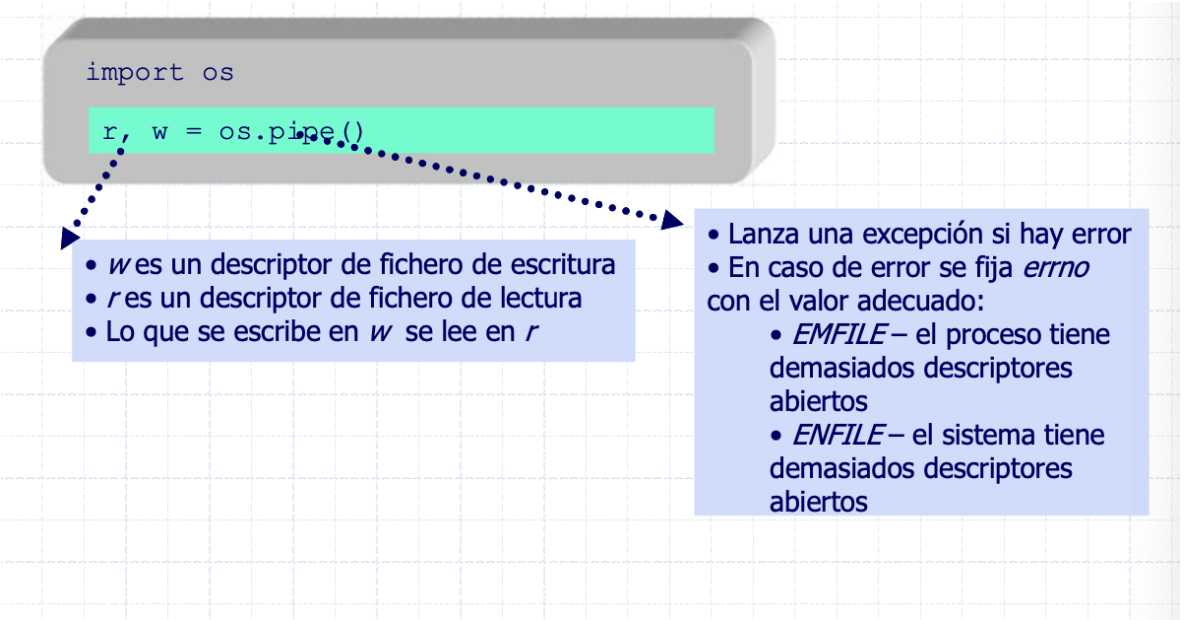
Acceso a las tuberías

- El acceso a las tuberías se realiza mediante descriptores de entrada/salida de unix
    - los mismos que devuelve la llamada open()
- La lectura/escritura se realiza mediante un chorro de bytes sin ninguna estructura
    - la lectura de los datos es independiente de la escritura
    - Permite leer de una vez de datos escritos en varias ocasiones
- Al realizar una escritura write() en una tubería...
    - si hay espacio suficiente, se escriben los bytes y la llamada retorna de inmediato

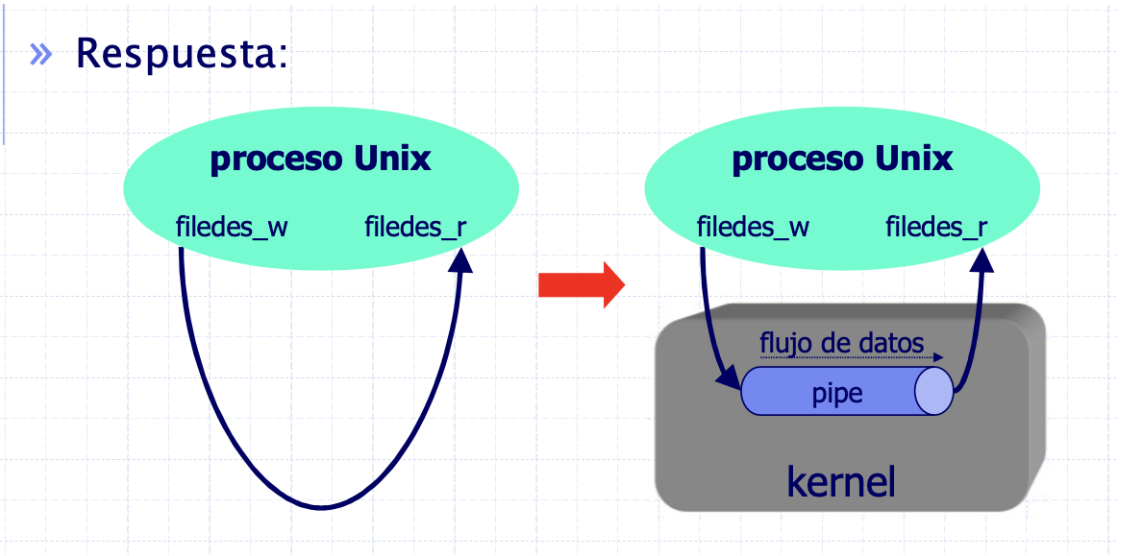
- Si no hay espacio suficiente, la llamada queda bloqueada y la ejecución del proceso es suspendida hasta que otro proceso haga sitio (leyendo datos de la tubería)

Creación de una tubería

- La llamada al sistema pipe() sirve para crear una tubería



La llamada pipe() crea dos descriptores de fichero

- 

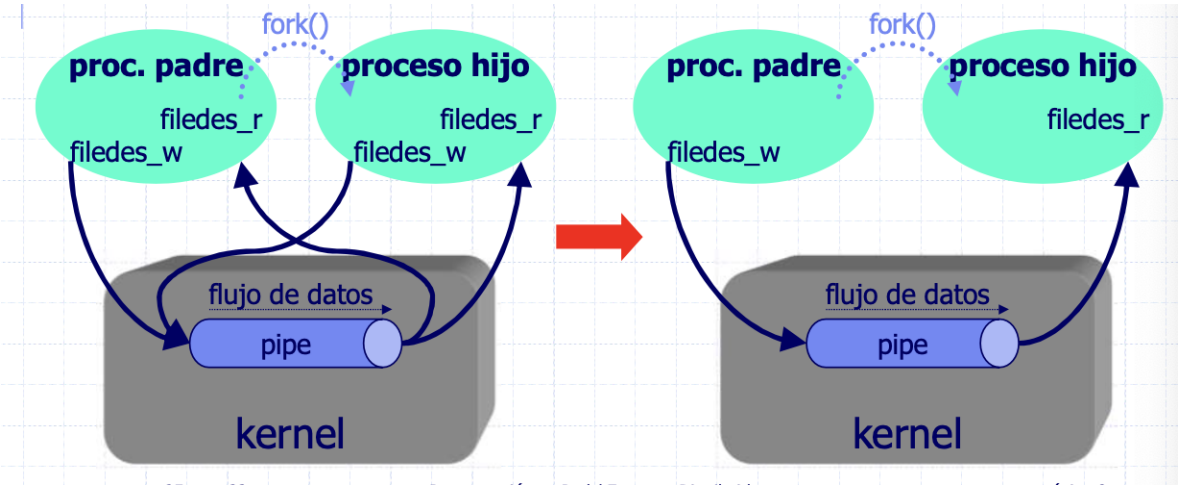
Dos extremos, no usar los mismos descriptores trae problemas

Conceptos básicos de las tuberías

- El tamaño del buffer de una tubería es finito, es decir, sólo pueden escribirse una cierta cantidad de bytes en la tubería (hasta que no se lean)
    - El tamaño máximo fijo para el buffer es típicamente de 512 bytes, que es el tamaño mínimo definido por POSIX
        - 512 bytes
            - sector de un disco
            - medio k
            - 1024/2
                - 512
    - Una ventaja de esto es que los datos raramente llegan a escribirse en el disco, sino que quedan en memoria (en la block buffer cache)
- Las tuberías sólo pueden ser usadas entre procesos que tienen un padre común
    - mitosis
- Un proceso creado con fork () hereda todas las tuberías abiertas que tenga su padre
    - Datos son los mismos
        - tabla de descriptores la heredo
        - hijo hereda las tablas de descriptores
            - 0 - in
            - 1 - out
            - 2 - error
    - códigos es igual

Procesos y tuberías

- Normalmente un proceso crea una tubería y luego hace un fork()
    - si se quiere comunicación de padre a hijo, se cierra el descriptor de lectura del padre y el de escritura del hijo
    - Si se quiere comunicación de hijo a padre, se cierra el descriptor de escritura del padre y el de lectura del hijo



- En el fork no se clona la tubería ya que la tubería es del kernel

fork() no tiene parametros

- exec permite cambiar tu memoria

Read devuelve los bytes que has leido

- cuando das un read y devuelve 0 es que termino

Entrada / Salida en las tuberías

- La tubería usa un buffer
  - Cuando el buffer está lleno, write(fd_w,) se bloquea
  - cuando el buffer esta vacío, read(fd_r) se bloquea
  - Si se intenta escribir cuando el extremo lector ha cerrado se genera SIGPIPE
  - cuando se cierra el extremo escritor, se recibe un EOF (end-of-file) tras la recepción de los último datos
- Para comunicación full-duplex tendríamos que usar dos tuberías
- Otra posibilidad útil
  - el hijo hace un exex() para ejecutar un programa
  - Pero antes el hijo conecta fd_r a stdin con lo que el padre puede enviar datos a la entrada standard del programa

Si el escritor cierra su extremo cierra la tubería
cuando lee le devuelve 0 el también tiene que cerrar

Señal

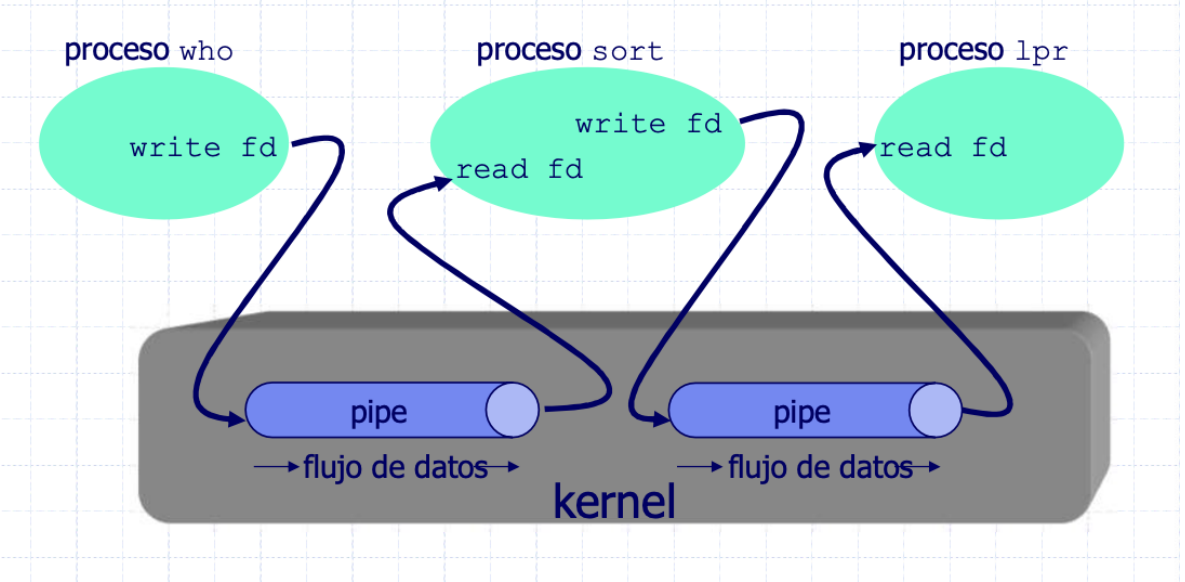
- cuando le mandan una señal a un proceso unix se muere

A que velocidad va esto entre dos procesos

- a la mas lenta
- el lee cuando quiere
- yo escribo a la velocidad que quiera si voy muy rápido me ponen a dormir
- yo leo a la velocidad que quiera si voy muy rápido me ponen a dormir

Tuberías en la shell de Unix

- La shell de unix usa tuberías y permite que el usuario las manipule
  - ejemplo who | sort | lpr



Riesgos de seguridad

- ninguno
- las tuberías solo la pueden usar mis hijos
- no hay forma de interceptar una tubería

Tuberías son como ficheros
Más grande el buffer más tarde te das cuenta que tienes un problema

---

FIFOs

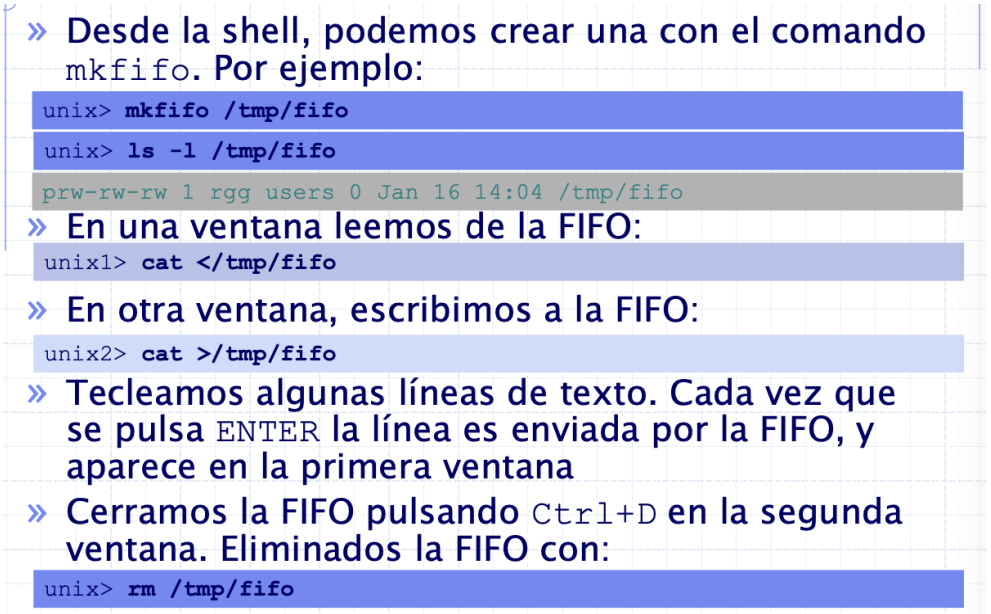
- El método de creación y uso de tuberías clásicas es muy restrictivo
- Para paliar este problema aparecen en 1982 las FIFOs o tuberías con nombre (nemed pipes)
- Utilizan un nombre en el sistema de ficheros:
  - Una FIFO es un fichero de tipo especial S_IFIFO
  - Se crean mediante las llamadas mkfifo () ó mknod()

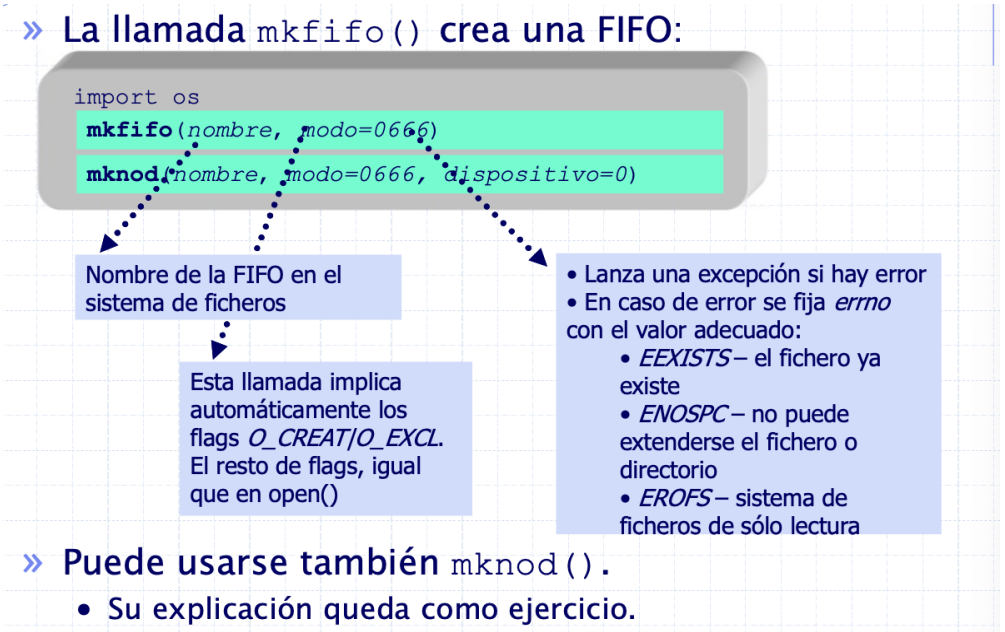
- Características principales
  - Permiten comunicación entre dos procesos cualesquiera aunque no estén relacionados
    - Cada proceso abre la FIFO con el modo que estime oportuno y realiza las operaciones necesarias
  - Son persistentes (sobreviven al proceso que las crea)

Los permisos de ficheros de unix

- sistema de protección
  - evitar hacernos daño
  - hola\n
  - mkfifo f
  - od -x hola
  - 5 bytes
  - echo hola > hola
  - echo hola >> hola
    - append
  - cat < f
    - activar extremo lector
    - escucha
  - cat > f
    - puedo escribir
    - activo extremo escritor
  - control d
    - salir sin matar al proceso
  - No te deja salir tienes que esperar que se conecte para escuchar
  - Si abrimos mas de un lector se van turnando
    - como lee uno el planificador le da acceso al otro
    - cambia de estado entonces sales de la cola FiFo
  - puedo tener dos escritores y un lector
    - escribo y lo recibe en cualquiera de las dos terminales
  - mientras hay alguien escribiendo el lector no termina
  - cachitos dependen del tamaño de la tubería
  - la tuberia siempre pesa 0
    - ocupa 0 bytes
      - parece un fichero pero no es un fichero
    - no ocupa ningun tamaño en memoría
    - no tiene tamaño en el disco

        - va por la memoria del kernel
        - no esta físicamente en el disco
    - ls -lai f
        - tiene un inode
            - guarda información del fichero donde esta en memoría, permisos etc
            - información
            - los ficheros nos tienen nombre
                - los nombres estan en los directorios
    - la mayor parte del tiempo los procesos estan dormidos
        - ps -a
            - R - running
            - S - sleep / suspended
        - ps aux





- Enlace simbolicos es fichero especial
  - d - directorio
  - p - pipe
  - - - fichero
  - El númerito que aparece es el link count
    - enlaces
  - El mismo inode mismo fichero
  - ln hola adios
  - ln -s
    - tiene inode distinto
  - dispositivo son ficheros especiales
    - c caracter
    - b bloque

Procesos y FiFos

- Una fifo debe tener al menos un proceso lector y un escritor
  - Puede haber muchos lectores y escritores, pero hay que tener en cuenta los posibles solapes en la escrituras
- La constante PIPE_BUF define el número máximo de caracteres que se pueden escribir en una FIFO atómaticamente
  - PIPE BUF = 1 K
  - Mando = 2 K
    - envio 1 K me duermo hasta que lea el 1 K y luego puedo volver a mandar

- Su funcionamiento depende de flag O_NONBLOCK
    - Si no se especifica un open() de sólo lectura se bloquea hasta que otro proceso abra el FIFO para escritura y viceversa
    - Si se especifica, un open () de sólo lectura retorna inmediatamente.
        - Un open() de sólo escritura retorna un errror si ningún proceso tiene la FIFO abierta para lectura

Precauciones a tomar con las FIFOs

- Crear una FIFO y abrirla para lectura y escritura involucra tres llamadas al sistema
    - La llamada pipe() hace lo mismo de una sola vez
- Los siguiente casos son llamadas al sistema bloqueantes, por regla general
    - Un proceso abre una FIFO en modo de solo lectura sin que existan otros procesos escritores
    - Un proceso abre una FIFO en modo de solo escritura sin que existan otros processo de lectores
- No se pueden abrir en modo lectura/escritura
    - lo que escribes lo vas a leer no tiene sentido
        - boca culo
    - si queremos bidireccional
        - dos FIFOs

3 llamadas

- mkfifo
- open lector
- open escritor

Chorros de bytes y mensajes

- Modelos de E/S para la transferencia de datos en canales punto a punto
    - como FIFOs y pipes
        - Chorro de bytes (byte stream)
            - no existe delimitadores
            - el receptor no puede saber si los datos que recibe fueron escritos de una vez o con michas operaciones write()
    - Mensajes - dos posibilidades
        - Uso de una estructura común para el mensaje, cuyo formato es conocido tanto por el emisor como el receptor

        - Requiere acuerdo previo entre ambos, generalmente en forma de código compartido
    - Uso de delimitadores de fin de mensaje
        - solo requiere acuerdo en el carácter delimitador

Delimitador \n