Santos Méndez Ulises Jesús 2CV15

Transferencia de datos entre procesos

El mecanismo de transferencia de datos pipe (canalización) estándar de UNIX permite que un proceso hijo herede el canal de comunicación de su padre; los datos escritos en un extremo del pipe pueden leerse en el otro.

Los pipes aparecen como otro tipo más de inodo en el software del sistema de archivos virtual.

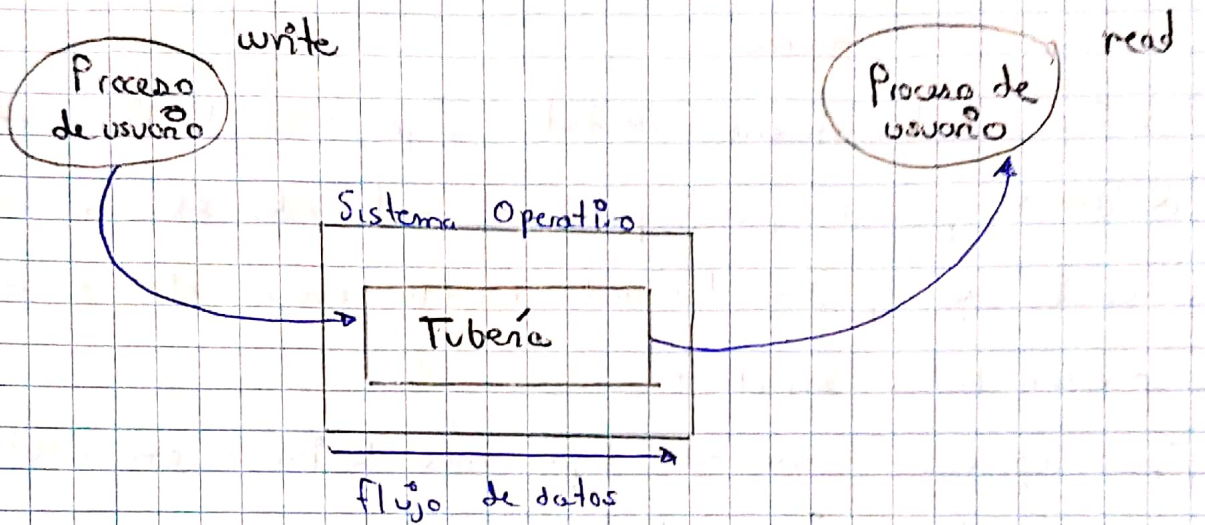
Cada pipe tiene un par de colas de espera para sincronizar al lector y al escritor.

Una tubería es un buffer circular que permite que dos procesos se comuniquen siguiendo el modelo productor-consumidor. Por tanto, se trata de una cola de tipo FIFO, en la que escribe un proceso y lee otro.

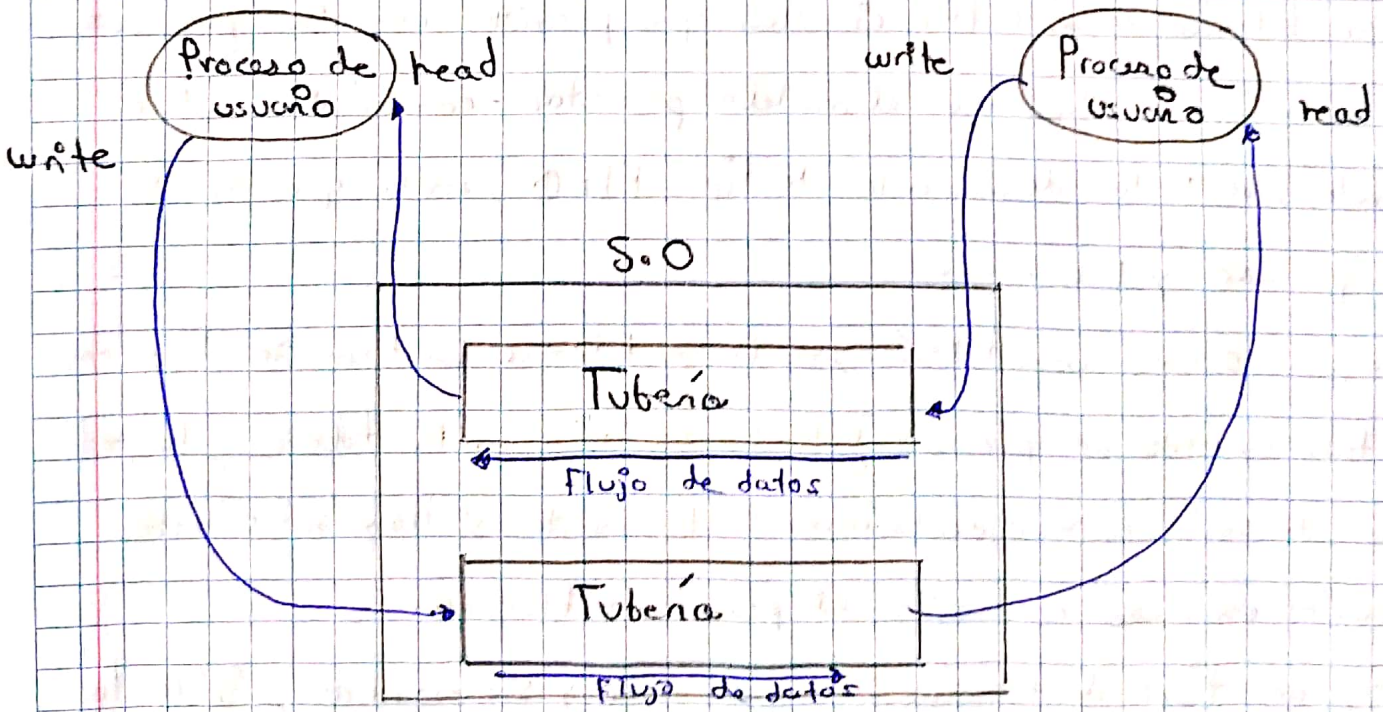
Cuando se crea una tubería, se le establece un tamaño fijo en bytes. Cuando un proceso intenta escribir en la tubería, la petición de escritura se ejecuta inmediatamente si hay suficiente espacio en caso contrario el proceso se bloquea.

Hay dos tipos de tuberías: con nombre y sin nombre. Sólo los procesos relacionados pueden compartir tuberías sin nombre, mientras que los procesos pueden compartir tuberías con nombre

tanto si están relacionados como si no.



Cuando se desea disponer de un flujo de datos bidireccional es necesario crear dos tuberías.



Una tubería es un mecanismo de comunicación con almacenamiento. El tamaño de una tubería varía en cada sistema operativo, aunque el tamaño típico es de 4KB. Sobre una tubería puede haber múltiples procesos lectores y escritores. Las tuberías se implementan normalmente como regiones de memoria compartida entre los procesos que utilizan la tubería.

Escritura en una tubería

Una operación de escritura sobre una tubería introduce datos en orden FIFO en la misma. La semántica de esta operación es la siguiente:

- Si la tubería se encuentra llena o se llena durante la escritura, la operación bloquea al proceso escritor hasta que se pueda completar.
- Si no hay ningún proceso con la tubería abierta para lectura, la operación devuelve el correspondiente error.
- Una operación de escritura sobre una tubería se realiza de forma atómica, es decir, si dos procesos intentan escribir de forma simultánea en una tubería, sólo uno de ellos lo hará. El otro se bloquea hasta que finalice la primera escritura.

Lectura de una tubería

Una operación de lectura de una tubería obtiene los datos almacenados en la misma. Las operaciones de lectura siguen la siguiente semántica:

- Si la tubería está vacía, la llamada bloquea al proceso en la operación de lectura hasta que algún proceso escriba datos en la misma.
- Si la tubería almacena M bytes y se quieren leer n bytes, entonces:
  - Si $M \geq n$, la llamada devuelve n bytes y elimina de la tubería los datos solicitados.
  - Si $M < n$, la llamada devuelve M bytes y elimina los datos disponibles en la tubería.
- Si no hay escritores y la tubería está vacía, la operación devuelve fin de archivo
- Al igual que las escrituras, las operaciones de lectura sobre una tubería son atómicas.

En general, la atomicidad en las operaciones de lectura y escritura sobre una tubería se asegura siempre que el número de datos involucrados en las anteriores operaciones sea menor que el tamaño de la misma.

Existen dos tipos de tuberías: sin nombre y con nombre. Una tubería sin nombre solamente se puede utilizar entre los procesos que desciendan del proceso que creó la tubería. Una tubería con nombre se puede utilizar para comunicar y sincronizar procesos independientes.

Existen tres formas, en general, de identificar a un mecanismo de comunicación o sincronización:

- Sin nombre: en este caso el servicio no tiene un nombre asociado y, por tanto, sólo puede ser utilizado por el proceso que lo crea y por aquellos procesos que lo heredan.
- Con un nombre local: en este caso el mecanismo tiene un nombre al que pueden acceder todos los procesos que ejecutan en la misma máquina y tengan permisos de acceso. Esto permite comunicar y sincronizar a procesos que residan en la misma máquina siempre que conozcan el nombre dado al mecanismo de comunicación o sincronización y tengan permisos para su utilización.
- Con nombre de red: en este caso el mecanismo tiene un nombre que lo identifica de forma única dentro de una red de computadoras y, por tanto, permite comunicar y sincronizar a procesos que ejecuten en computadoras distintas.

- A comparación de los pipes con el mecanismo por comunicación por mensajes es que el de mensajes tiene una mayor ventaja ya que UNIX proporciona llamadas a sistema msgsnd y msgrcv para que los procesos reciban y transfieran mensajes, actuando como buzón.
- La memoria compartida tiene cierta ventaja sobre los pipes ya que la memoria virtual se comparte para muchos procesos. Los procesos leen y escriben en la memoria compartida utilizando las mismas instrucciones máquina y el proceso puede ser de solo lectura o escritura estableciéndose de forma individual.
- Los pipes tienen ventaja sobre los semáforos ya que en los semáforos requiere un valor actual para el semáforo, un identificador del último proceso, el número de procesos en espera y se asocia una cola de procesos (bloqueados), requiere una serie de llamadas a sistema.
- Los pipes tienen ventaja sobre las señales ya que pueden tener un orden, en cambio, las señales no ya que se tratan por igual, un proceso realiza una acción por defecto no demás para responder la señal.

Un pipe no tiene nombre y por lo tanto solo puede ser utilizado entre los procesos que lo heredan a través de la llamada fork().

Creación de un pipe

El servicio que permite crear un pipe es el siguiente:

int pipe (int fd[2]); Esta llamada devuelve dos descriptores de archivos que se utilizan como identificadores.

fd[0], descriptor de archivo que se emplea para leer del pipe

fd[1], descriptor de archivo que se utiliza para escribir en el pipe

La llamada pipe devuelve 0 si fue bien y -1 en caso de error.

Escritura en un pipe

int write (int fd, char * buffer, int n);

Lectura en un pipe

int read (int fd, char * readbuffer, int n);

Cierre de un pipe

El cierre de cada uno de los descriptores que devuelve la llamada pipe se consigue mediante close.

int close (int fd);

La llamada devuelve 0 si se ejecuto con éxito. En caso de error devuelve -1.