MÓDULO II. Uso de los Servicios del SO Linux mediante la API

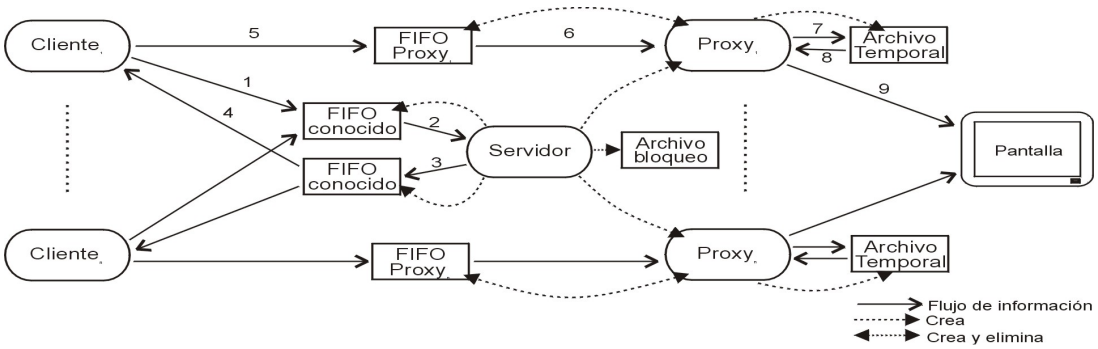
Sesión 7. Construcción de un spool de impresión

19/07/2025

Índice

1. Enunciado del ejercicio

Esta práctica tratará aspectos relacionados con **procesos**, **señales** y **cauces con nombre** (archivos FIFO). El ejercicio plantea la programación de un *spool* concurrente de impresión en pantalla. Su funcionamiento general consiste en controlar el acceso de procesos clientes al recurso compartido, en este caso la pantalla, garantizando la utilización en exclusión mutua de dicho recurso. Un sistema *spool* para impresión imprime un documento sólo cuando éste se ha generado por completo. De esta forma, se consigue que un proceso no pueda apropiarse indefinidamente, o durante mucho tiempo si el proceso es lento, del recurso compartido. Como mecanismo de comunicación/sincronización entre procesos se van a utilizar cauces con nombre (archivos FIFO) y en algún caso señales. En la siguiente figura se muestra el esquema general a seguir para la implementación:



Siguiendo los mensajes numerados, obtendremos las interacciones entre procesos para poder llevar a cabo la impresión de un archivo, según se explica a continuación:

1. Un cliente solicita la impresión de un archivo enviando un mensaje (cuyo contenido no tiene importancia) al servidor a través de un **FIFO** cuyo nombre es conocido.
2. El servidor lee esta petición delegando la recepción e impresión del documento en un proceso (*proxy*) que crea específicamente para atender a dicho cliente. Una vez servida dicha petición, el *proxy* terminará.
3. El servidor responde al cliente a través de otro **FIFO** de nombre conocido, informando de la identidad (**PID**) del *proxy* que va a atender su petición. Este dato es la base para poder comunicar al cliente con el *proxy*, ya que éste creará un nuevo archivo **FIFO** específico para esta comunicación, cuyo nombre puede

ser el propio PID del *proxy*. El *proxy* se encargará de eliminar dicho **FIFO** cuando ya no sea necesario (justo antes de terminar su ejecución).
4. El cliente lee esta información que le envía el servidor, de manera que así sabrá dónde enviar los datos a imprimir.
5. Probablemente el cliente necesitará enviar varios mensajes como éste, tantos como sean necesarios para transmitir toda la información a imprimir. El final de la transmisión de la información lo indicará con un fin de archivo.
6. El *proxy* obtendrá la información a imprimir llevando a cabo probablemente varias lecturas como ésta del **FIFO**.
7. Por cada lectura anterior, tendrá lugar una escritura de dicha información en un archivo temporal, creado específicamente por el *proxy* para almacenar completamente el documento a imprimir.
8. Una vez recogido todo el documento, volverá a leerlo del archivo temporal justo después de comprobar que puede disponer de la pantalla para iniciar la impresión en exclusión mutua.
9. Cada lectura de datos realizada en el paso anterior implicará su escritura en pantalla.

Tenga en cuenta las siguientes consideraciones de cara a la implementación:

- Utilice un tamaño de 1024 bytes para las operaciones de lectura/escritura (mediante las llamadas al sistema `read`/`write`) de los datos del archivo a imprimir.
- Recuerde que cuando todos los procesos que tienen abierto un **FIFO** para escritura lo cierran, o dichos procesos terminan, entonces se genera automáticamente un fin de archivo que producirá el desbloqueo del proceso que esté bloqueado esperando leer de dicho **FIFO**, devolviendo en este caso la llamada al sistema `read` la cantidad de 0 Bytes leídos. Si en algún caso, como por ejemplo en el servidor que lee del **FIFO** conocido no interesa este comportamiento, entonces la solución más directa es abrir el **FIFO** en modo lectura/escritura (`O_RDWR`) por parte del proceso servidor. Así nos aseguramos que siempre al menos un proceso va a tener el **FIFO** abierto en escritura, y por tanto se evitan la generación de varios fin de archivo.
- Siempre que cree un archivo, basta con que especifique en el campo de modo la constante `S_IRWXU` para que el propietario tenga todos los permisos (lectura, escritura, ejecución) sobre el archivo. Por supuesto, si el sistema se utilizara en una situación real habría que ampliar estos permisos a otros usuarios.
- Utilice la función `tmpfile` incluida en la biblioteca estándar para el archivo temporal que crea cada *proxy*, así su eliminación será automática. Tenga en cuenta que esta función devuelve un puntero a la estructura `FILE` (`FILE *`), y por tanto las lecturas y escrituras se realizarán con las funciones de la biblioteca estándar `fread` y `fwrite` respectivamente.
- No deben quedar procesos zombis en el sistema. Podemos evitarlo manejando en el servidor las señales `SIGCHLD` que envían los procesos *proxy* (ya que son hijos del servidor) cuando terminan. Por omisión, la acción asignada a esta señal es ignorarla, pero mediante la llamada al sistema `signal` podemos especificar un manejador que ejecute la llamada `wait` impidiendo que los procesos *proxy* queden en estado zombi indefinidamente.
- Por cuestiones de reusabilidad, el programa *proxy* leerá de su entrada estándar (constante `STDIN_FILENO`) y escribirá en su salida estándar (constante `STDOUT_FILENO`). Por tanto, hay que redireccionar su entrada estándar al archivo **FIFO** correspondiente. Esto se puede llevar a cabo mediante la llamada al sistema `dup2`.
- Para conseguir el bloqueo/desbloqueo de pantalla a la hora de imprimir, utilice las funciones vistas en la sesión 6 para bloqueo de archivos sobre un archivo llamado `bloqueo` que será creado en el proceso servidor.

También se proporciona un programa denominado `clientes.c` capaz de lanzar hasta 10 clientes solicitando la impresión de datos. Para simplificar y facilitar la comprobación del funcionamiento de todo el sistema, cada uno de los clientes pretende imprimir un archivo de tamaño desconocido pero con todos los caracteres idénticos, es

decir, un cliente imprimirá sólo caracteres a, otro sólo caracteres b, y así sucesivamente. El formato de ejecución de este programa es:

`clientes <nombre_fifos_conocidos> <número_clientes>`

El argumento `<nombre_fifos_conocidos>` es un único nombre, de forma que los clientes suponen que el nombre del **FIFO** conocido de entrada al servidor es dicho nombre concatenado con el carácter “e”. En el caso del **FIFO** de salida, se concatena dicho nombre con el carácter “s”.

Implemente el resto de programas según lo descrito, para ello necesitará además de las funciones y llamadas al sistema comentadas anteriormente, otras como: `mkfifo` (crea un archivo **FIFO**), `creat` (crea un archivo normal), `unlink` (borra un archivo de cualquier tipo).

Ten en cuenta que el servidor debe ser un proceso que está permanentemente ejecutándose, por tanto, tendrás que ejecutarlo en *background* y siempre antes de lanzar los clientes. Asegúrate también de que el servidor cree los archivos **FIFO** conocidos antes de que los clientes intenten comunicarse con él.