TALLER PIPES SISTEMAS OPERATIVOS

1) Tablas

- Tabla de descriptores: Cada que se usa open busca una entrada libre, y cuando se encuentra esta entrada se enlaza con la tabla de archivos global. Su propósito es realizar un seguimiento de los descriptores y sus propiedades, como lo son el estado, ubicación y permisos. Esta se almacena a nivel de proceso.
- Tabla de archivos global: El propósito es mirar el numero de entradas que apuntan, como se abrió el archivo y el offset. Adicionalmente su función principal es permitir un acceso eficiente y coordinado a los archivos compartidos entre múltiples procesos. Esta se almacena a nivel de sistema operativo.
- Tabla de inodos: Tiene toda la información de un archivo, creando un inodo con los metadatos de este archivo, siendo esta su función principal, almacenar los metadatos, entre estos metadatos están la ubicación de los bloques de datos, los permisos de acceso, la fecha de creación y modificación, etc. Esta tabla se almacena a nivel de sistema de archivos, o disco.

2) Descriptor de archivos

Es un identificador numérico que se utiliza para acceder a archivos y otros recursos de entrada/salida, como lo son los sockets y los pipes. Adicionalmente estos descriptores son usados por los programas para leer, escribir y realizar otras acciones de entrada/salida en archivos y dispositivos.

3) Llamada al sistema DUP

La llamada al sistema dup sirve para duplicar un descriptor de archivo ya existente. Su principal utilidad es crear una nueva entrada en la tabla de descriptores de archivos del proceso que hace referencia al mismo archivo o recurso que el descriptor de archivo original.

4) Redireccionar salida estándar de un proceso a un archivos

Linux:

```
proceso > salida.txt
```

C:

```
#include <stdio.h>
#include <stdlib>
#include <unistd.h>
#include <fcntl.h>

int main(){
	int file_descriptor = open(“salida.txt”, O_WRONLY | O_CREAT, 0666);

	if (file_descriptor == -1){
		perror(“Error al abrir el archivo\n”);
		exit(1);
	}

	if(dup2(file_descriptor, 1) == -1){
		perror(“Error al redirigir la salida estandar\n”);
		exit(1);
	}

	printf(“Mensaje redirigido por salida estandar a salida.txt”);
	close(file_descriptor);

	return 0;
}
```

5) Pipes
Los pipes son mecanismos de comunicación en sistemas operativos , estos permiten la comunicación e intercambio de datos entre dos procesos en tiempo real. Los pipes establecen canales unidireccionales de comunicación entre los procesos, donde un proceso actúa como escritor y otro como lector.

6) Pipes nominales y no nominales
Los pipes no nominales son temporales, estos existen mientras los procesos que los utilizan están en ejecución. Cuando los procesos mueren, el pipe desaparece, mientras que los pipes nominales son persistentes y existen en el sistema de archivos incluso después de cerrar los procesos que los usan. Adicionalmente los pipes no nominales son unicamente unidireccionales, mientras que los nominales pueden ser bidireccionales.
Otra diferencia importante es el hecho de que los pipes no nominales no tienen nombre, haciendo así que no puedan ser accedidos desde el sistema de archivos limitando su creación y utilización exclusivamente de forma directa en la memoria del sistema. Por otro lado los pipes nominales al tener nombre asociado permite que múltiples procesos lo puedan usar refiriéndose a su nombre

7) ¿Que sucede en el código?
Int myfd[2];
pipe(myfd);

1° paso: Se crea un array de enteros de dos posiciones, en donde posteriormente vamos a guardar los descriptores de archivos asociados con el pipe, donde la posición 0 será usada para la lectura y la 1 para la escritura del pipe
2° paso: Se realiza una llamada al sistema “pipe()” el cual crea un pipe en el sistema operativo y asocia los extremos con los descriptores de archivos, guardándolos en myfd como se explicó anteriormente. Este pipe creado es no nominal, por lo tanto no tiene nombre en el sistema de archivos
3° paso: El sistema operativo crea una estructura de datos especial para representar el pipe. Esta estructura almacena información como el buffer de datos del pipe, los extremos de lectura y escritura, etc.
4° paso: Se asignan recursos en el sistema operativo para almacenar el buffer de datos del pipe.

8) Leer pipes vacíos
Cuando un proceso intenta leer un pipe vacío primero se bloquea el proceso lector en el extremo de lectura del pipe, posterior a esto se queda en estado de espera hasta que otro proceso escriba datos en el otro extremo del pipe, cuando este proceso escritor escribe datos, estos se almacenan en el buffer del pipe y por ultimo cuando los datos se terminan de almacenar en el buffer el proceso lector lee los datos del buffer,

9) Escribir en un pipe cerrado
Sin un proceso intenta escribir en un pipe cerrado se produce un error, indicando que ha fallado el intento de escritura.

10) Pipe nominales
creación:
#include <sys/types.h>
#include <sys/stat.h>

int mkfifo(const char *pathname, mode_t mode);

No es necesario que todos los procesos que se comunican creen el pipe nominal, generalmente se crea por adelantado por un proceso y se comparte por nombre.

11) ¿Que procesos deben abrir el pipe nominal?
No hay una restricción sobre que procesos pueden abrir un pipe nominal. Cualquier proceso puede abrir estos pipes para lectura y escritura siempre y cuando tengan los permisos necesarios. Usualmente los casos que requieren abrir pipes nominales son cuando se requiere comunicar procesos de usuario, comunicar procesos y procesos del sistema, comunicación entre procesos remotos, etc...