```
1   /*              tarea7nueva.c
2
3   Programa ilustrativo del uso de pipes y la redirecciÛn de entrada y
4   salida est·ndar: "ejemploDirectorio | wc" */
5   #include<sys/types.h>
6   #include<fcntl.h>
7   #include<unistd.h>
8   #include<stdio.h>
9   #include<stdlib.h>
10  #include<errno.h>
11  int main(int argc, char *argv[])
12  {
13  int fd[2];
14  pid_t PID;
15  pipe(fd); // Llamada al sistema para crear un pipe
16  if ( (PID= fork())<0) {
17                  perror("fork");
18                  exit(1);
19  }
20  if(PID == 0) { // ------------ hijo
21                  close(fd[0]);//cierro lo que no uso
22                  close(STDOUT_FILENO);
23                  dup(fd[1]);
24                  execlp("./ejemploDirectorio","ejemploDirectorio",argv[1], NULL);
25  }
26  else {
27                  close(fd[1]); //cierro lo que no uso
28                  close(STDIN_FILENO);
29                  dup(fd[0]);
30                  execlp("wc","wc",NULL);
31  }
32  return(0);
33  }
```

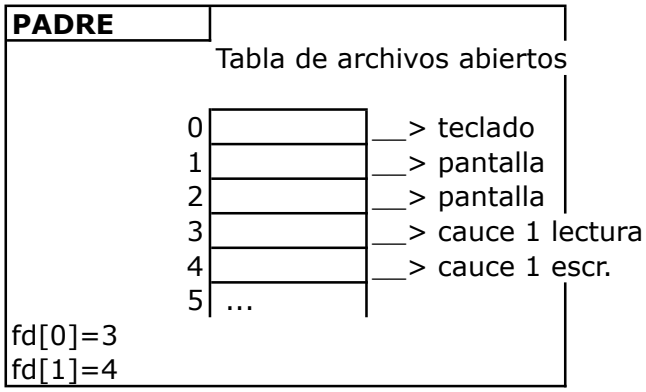
Archivo ejemploDirectorio:

```
// programa que muestra los nombres de los hijos dle directorio dado como argumento
#include <stdio.h>
#include <stdlib.h>
#include <dirent.h>
int main( int argc, char *argv[] )
{       DIR *dir;
        struct dirent *mi_dirent;
        if( argc != 2 )  {
                printf( "Debe dar un argumento");
                exit( -1 );  }
        if( (dir = opendir( argv[1] )) == NULL ) {
                perror( "opendir" );
                exit( -1 );  }
        while( (mi_dirent = readdir( dir )) != NULL )
                printf( "%s\n", mi_dirent->d_name );
        ///// programa que escribe en STDOUT_FILENO (=1)
        closedir( dir );
        return 0;
}
```

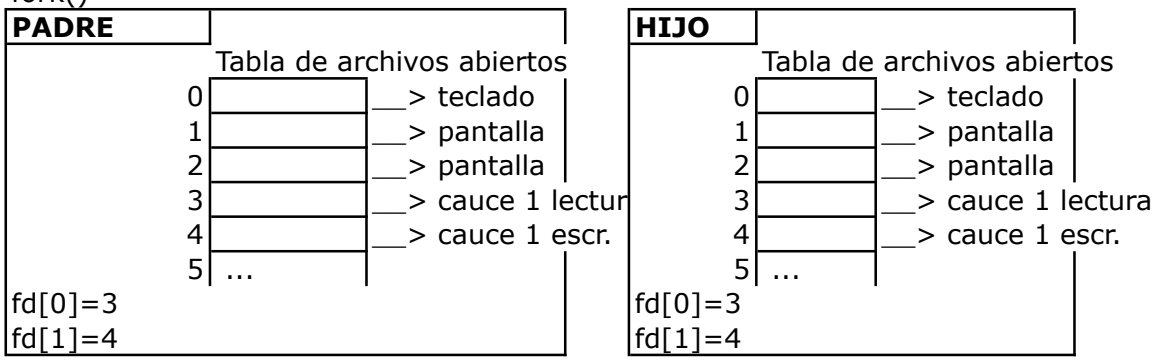
ATENCION:

A continuación se muestra el efecto de la ejecución de algunas instrucciones especialmente relevantes del programa tarea7nueva.c. Recordemos que los procesos padre e hijo que se lanzan son actividades concurrentes, cuyo orden de ejecución no es conocido de antemano. El orden en que se muestran las figuras a continuación no presupone que estemos pensando en tal orden de ejecución.

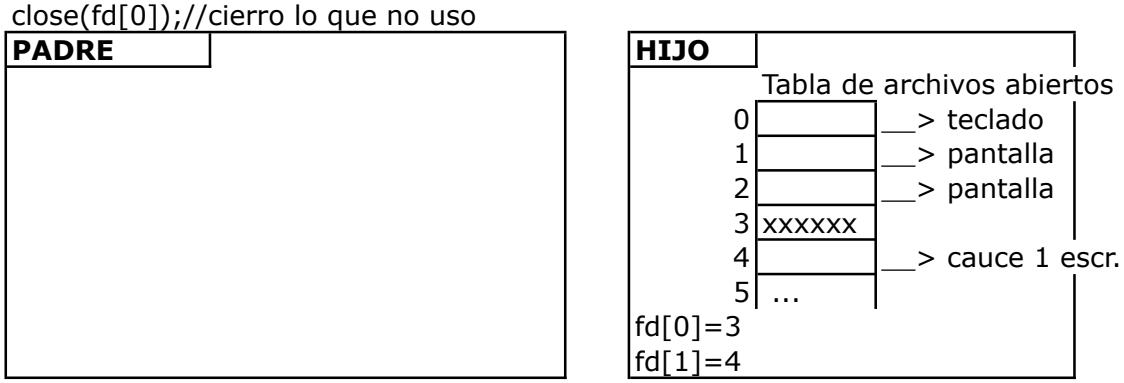
Tras 15:



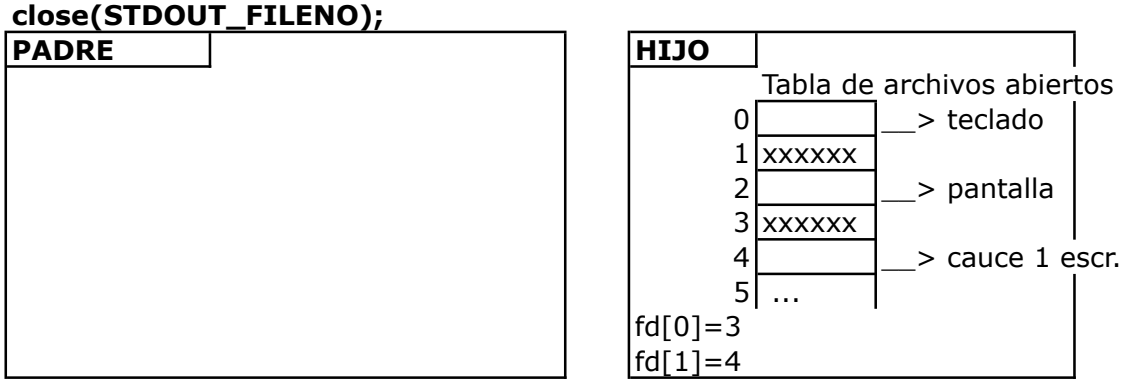
Tras 16: fork()



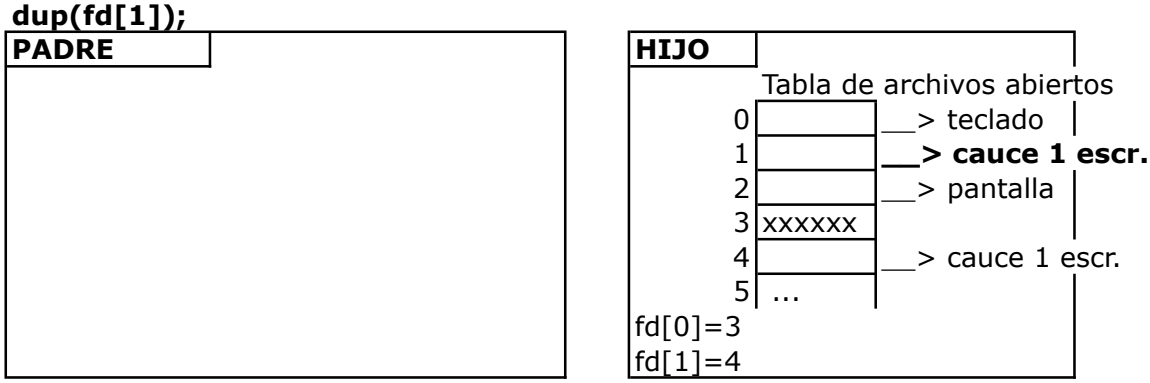
Tras 21: close(fd[0]);//cierro lo que no uso



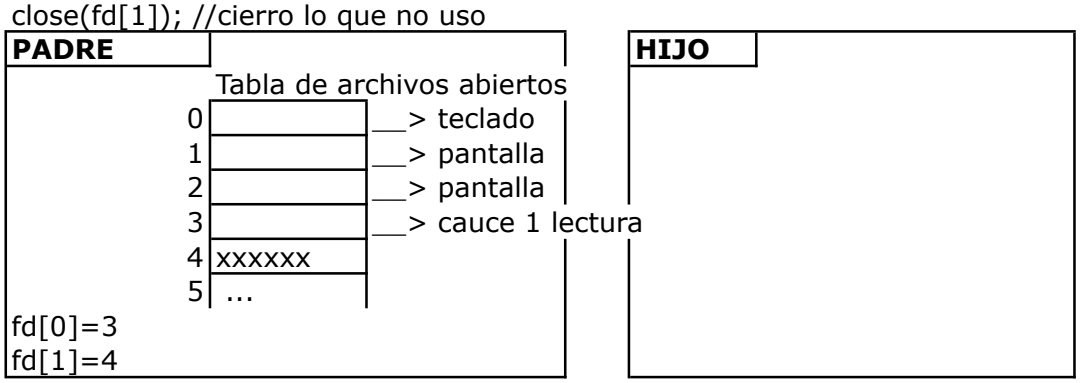
Tras 22: **close(STDOUT_FILENO);**



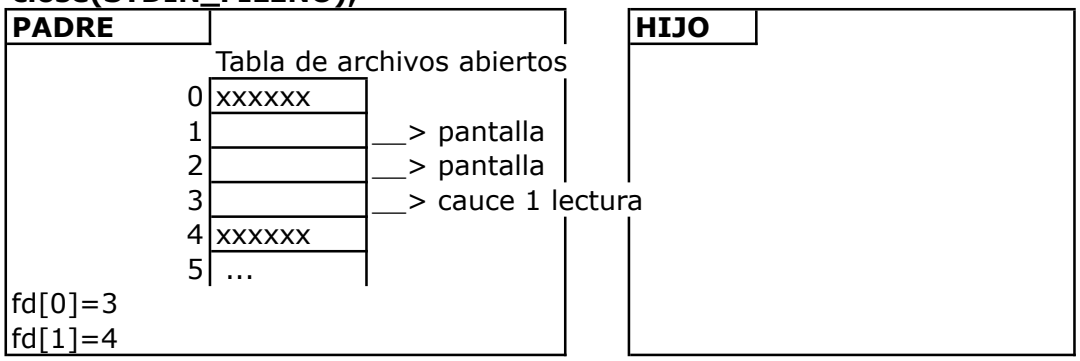
Tras 23: **dup(fd[1]);**



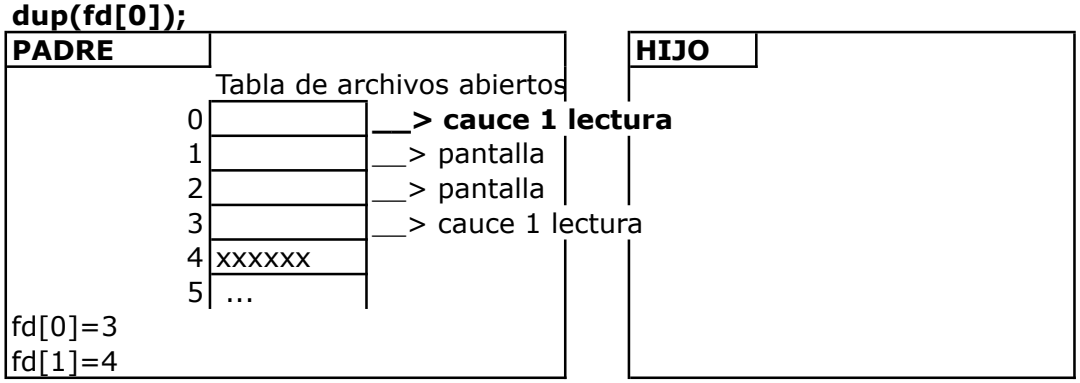
Tras 27: close(fd[1]); //cierro lo que no uso



Tras 28: close(STDIN_FILENO);



Tras 29: dup(fd[0]);



COMO RESULTADO DE LO ANTERIOR:

* Si en el hijo se escribe en 1, va a ir realmente al cauce
* Si en el padre se lee de 0, se lee realmente del cauce

Con ello hemos conseguido redirigir la salida estandar del hijo, y la entrada estandar del padre.
Sin cambiar el código de los programas que se lanzan con exec conseguimos que
escriban "en otro sitio" o lean "en otro sitio"