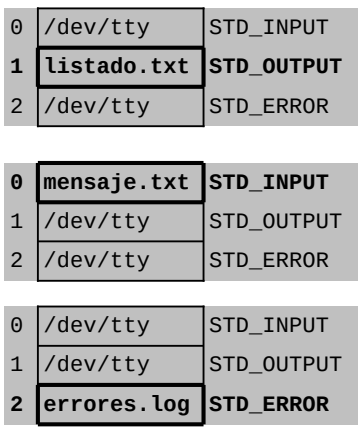
Redirecciones y Tuberías

Ampliación de Sistemas Operativos (prácticas)
E.U. Informática en Segovia
Universidad de Valladolid

Redirecciones

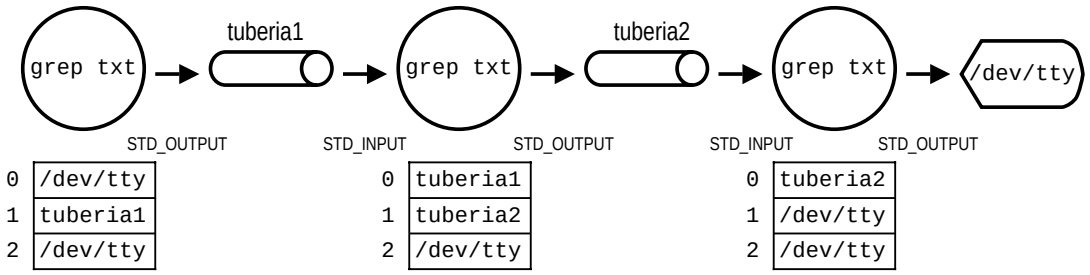
- Redirección de la entrada y la salida estándar
  - Unix/Linux definen tres descriptores con un significado especial, que suelen estar asociados al terminal
    - `0` `STD_INPUT`: entrada estándar de datos
    - `1` `STD_OUTPUT`: salida estándar de datos
    - `2` `STD_ERROR`: salida estándar de errores
  - Redirección de la salida estándar
    - `$ ls –la > listado.txt`
  - Redirección de la entrada estándar
    - `$ write fdiaz < mensaje.txt`
  - Redirección de la salida de error estándar
    - `$ cc hilos.c 2> errores.log`



Tuberías

- Comunicación de procesos Unix/Linux
  - La comunicación entre procesos en Unix/Linux puede realizarse mediante tuberías
  - Las tuberías son un tipo especial de ficheros de capacidad limitada con acceso secuencial
  - Las tuberías pueden compartirse (entre procesos relacionados mediante la relación padre-hijo) gracias al mecanismo de herencia

```
$ ls | grep txt | sort
```



Redirecciones y tuberías: llamadas al sistema

Redirecciones y tuberías	
dup2	Duplicar un descriptor de fichero
pipe	Creación de una tubería
mkfifo	Creación de una tubería con nombre (fifo)

Redirecciones: dup y dup2

- **dup, dup2**: Duplicar un descriptor de fichero

```
#include <unistd.h>

int dup (int fd)
int dup2 (int fd, int copia_fd)
```

- Descripción
  - **dup**: retorna un descriptor de fichero cuya entrada es una copia de la que se corresponde con el parámetro **fd**. El descriptor que retorna es el descriptor disponible más bajo
  - **dup2**: cierra el descriptor que corresponde a **copia_fd** y luego copia la entrada del correspondiente al primer parámetro **fd** sobre la del segundo parámetro **copia_fd**
- Valor de retorno
  - Nuevo descriptor del fichero
  - -1 en caso de error (descriptor no válido o se supera el máximo de ficheros abiertos, **OPEN_MAX**)

Redirecciones: Ejemplo

```
#include <sys/types.h>
#include <sys/stat.h>
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>

#define NEW (O_WRONLY|O_CREAT|O_EXCL)
#define PERM (S_IRUSR|S_IWUSR|S_IRGRP|S_IROTH)

int redirigir_sal (const char *fich) {
  int fd;

  if ( (fd=open(fich, NEW, PERM)) == -1 )
    return -1;
  if ( dup2(fd, STDOUT_FILENO) == -1)
    return -1;
  close(fd);
  return 0;
}
```

```
int main (int argc, char * argv[]) {
  int fd_origen, fd_destino;

  if (argc < 3) {
    fprintf(stderr, "Uso: %s fich_salida
                     orden args\n", argv[0]);
    exit(1);
  }

  if (redirigir_sal(argv[1]) == -1) {
    fprintf(stderr, "No se puede redirigir salida
                      hacia %s\n", argv[1]);
    exit(1);
  }

  if ( execvp(argv[2], &argv[2]) < 0) {
    fprintf(stderr, "no se puede ejecutar %s\n",
                     argv[2]);
    exit(1);
  }
}
```

Tuberías: pipe

- **pipe**: Creación de una tubería (en memoria)

```
#include <unistd.h>

int pipe (int desc[2])
```

- Descripción
  - **pipe**: crea una estructura de datos en memoria, denominado tubería e inicialmente vacía, que se maneja como una cola FIFO de bytes
  - Las tuberías son un mecanismos de comunicación entre proceso Unix/Linux
  - Tras realizar la llamada, **desc[0]** es un descriptor de fichero válido para leer de la tubería (salida), mientras que **desc[1]** es un descriptor de fichero válido para escribir en la tubería (entrada)
  - La capacidad máxima de una tubería está limitada por la implementación,
  - Las tuberías junto con los descriptores de acceso son heredados por los procesos hijos y preservados tras la ejecución de **exec**
- Valor de retorno
  - 0 si no hay error
  - -1 en caso de error (array de descriptores **desc** no válido o se supera el máximo de ficheros abiertos, **OPEN_MAX**)

Tuberías

- Lectura de una tubería
  - Una vez abierta una tubería, por ejemplo, mediante **pipe**, puede leerse de la tubería, utilizando la llamada al sistema **read**, con una serie de consideraciones especiales:
    - Si existen bytes disponibles, se leen como mucho los solicitados en la llamada a **read**
    - Si en la tubería no hay datos, **read** suspende al proceso que lo invoca hasta que existan datos en la tubería (se puede evitar esta situación utilizando el modo de lectura no bloqueante **O_NONBLOCK** mediante la llamada al sistema **fnctl**)
    - Cuando no existe ningún descriptor de escritura asociada a la tubería (tanto del proceso lector como de cualquier otro proceso) **read** no suspende al proceso y retorna el valor 0, indicando así la condición de **final de datos en la tubería** (final de fichero)

Tuberías

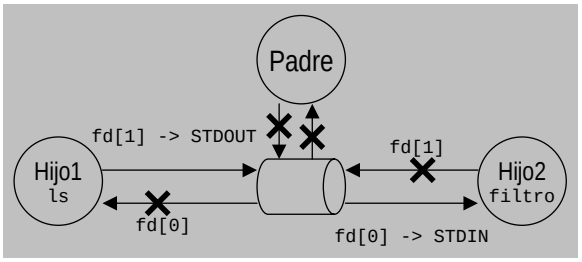
- Escritura en una tubería
  - Una vez abierta una tubería, por ejemplo, mediante **`pipe`**, puede escribirse en la tubería, utilizando la llamada al sistema **`write`**, con una serie de consideraciones especiales:
    - Si existe capacidad suficiente en la tubería para almacenar la cantidad de bytes solicitados en la llamada a **`write`**, éstos se almacenan en la tubería al final y, además, la operación puede considerarse atómica
    - Si no se pueden almacenar los datos de la escritura porque no hay suficiente sitio, el proceso escritor se bloquea hasta que exista disponibilidad de espacio
    - Si se intenta escribir en una tubería que no posee ningún descriptor de lectura asociado (tanto del proceso escritor como de cualquier otro proceso), el proceso que intenta escribir recibe la señal **`SIGPIPE`**. Este mecanismo facilita la eliminación automática de una cadena de procesos comunicados por tuberías cuando se aborta inesperadamente un componente de la cadena

Tuberías: Ejemplo 1

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/wait.h>

int main (int argc, char *argv[]) {
  int i, fd[2], pid[2];
  /* Comprobación argumentos */
  if (argc < 2) {
    fprintf(stderr, "Uso: %s filtro\n", argv[0]);
    exit(1);
  }
  /* Creación recurso tubería, por el padre */
  pipe(fd);
  /* 1er hijo, ejecuta ls */
  if ( (pid[0]= fork()) == 0) {
    dup2(fd[1], STDOUT_FILENO);
    close(fd[0]);
    close(fd[1]);
    execlp("/bin/ls", "ls", NULL);
    perror("Hijo1: Fallo al hacer exec");
    exit(1);
  }
```

```
  /* 2do hijo, ejecuta filtro */
  if ( (pid[1]=fork()) == 0) {
    dup2(fd[0], STDIN_FILENO);
    close(fd[0]);
    close(fd[1]);
    execvp(argv[1], &argv[1]);
    perror("Hijo2: Fallo al hacer exec");
    exit(1);
  }
  /* El padre no interviene */
  close(fd[0]);
  close(fd[1]);
  for (i=0; i<2; i++)
    wait(pid[i], NULL);
  exit(0);
}
```



Tuberías: Ejemplo 2

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/wait.h>
#define  TAM 256

int main (int argc, char *argv[]) {
  int fd[2], status;
  char buffer[TAM];
  unsigned lon;
  if (argc != 1) {
    fprintf(stderr,"Uso: %s\n",argv[0]);
    exit(1);
  }
  /* Creación recurso tubería */
  pipe(fd);
  switch (fork()) {
    case -1: fprintf(stderr, "Error al
                      hacer fork");
             exit(1);
             break;
    case 0: /* Código del hijo */
      dup2(fd[1], STDOUT_FILENO);
      close(fd[0]); close(fd[1]);
      fprintf(stderr, "Hijo[%ld] va a escribir en la
                       tuberia\n",(long)getpid());
      sprintf(buffer, "MENSAJE ESCRITO POR EL HIJO
                       [%ld]", (long)getpid());
      lon=strlen(buffer)+1;
      if ( write(1, buffer, lon) != lon ) {
        fprintf(stderr, "Fallo al escribir el hijo\n");
        exit(1);
      }
      break;
    default:/* Código del padre */
      dup2(fd[0], STDIN_FILENO);
      close(fd[0]); close(fd[1]);
      fprintf(stderr, "Padre[%ld] va a leer de la
                       tuberia\n",(long)getpid());
      while ( (wait(&status)==-1) && (errno==EINTR) );
      if ( read(0, buffer, TAM) <= 0) {
        fprintf(stderr,"Fallo al leer el padre\n");
        exit(1);
      }
    printf("Proceso PADRE (mensaje leido): %s\n", buffer);
    }
    exit(0);
}
```


Tuberías: mkfifo

- **`mkfifo`**: creación de una tubería con nombre o fifo

```
#include <sys/types.h>
#include <sys/stat.h>

int mkfifo (char *ruta, mode_t modo)
```

- Descripción
  - Crea un fichero especial de tipo fifo (o tubería con nombre)
  - El nombre (por ejemplo, **`/tmp/fifo_ida`**) se especifica en el argumento **`ruta`**, mientras que el parámetro **`modo`** establece la forma de creación de la tubería
  - Las fifos, a diferencia de las tuberías sin nombre, persisten aunque todos los procesos los hayan cerrado (se guardan en un volumen de disco)
  - Para acceder a una fifo, todos los procesos deben tener los permisos adecuados
- Valores de retorno
  - 0 si funciona con éxito
  - -1 en caso de error (no se tiene permiso de acceso, la fifo ya existe, etc.)

Tuberías: Ejemplo

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <sys/stat.h>
#define  TAM 256
int main(int argc, char *argv[]) {
  mode_t fifo_perm = S_IRUSR | S_IWUSR;
  int fd, status;
  char buffer[TAM];
  unsigned lon;
  if (argc != 2) {
    fprintf(stderr, “Uso: %s nombre_fifo\n”, argv[0]);
    exit(1);
  }
  if ( (mkfifo(argv[1], fifo_perm) == -1) && (errno != EEXIST) ) {
    fprintf(stderr, “No se puede crear la fifo: %s\n”, argv[1]);
    exit(1);
  }
```


Tuberías: Ejemplo

```
 switch ( fork() ) {
   case –1: /* Error al hacer fork */
     fprintf(stderr, “Error al hacer fork\n”);
     exit(1);
     break;
   case 0:  /* Código del proceso hijo */
     fprintf(stderr, “Hijo [%ld] va a abrir la fifo %s\n”, (long)getpid(),
                      argv[1]);
     if ( (fd=open(argv[1], O_WRONLY)) == -1) {
       fprintf(stderr, “El hijo no puede abrir la fifo %s\n”, argv[1]);
       exit(1);
     }
     sprintf(buffer, “MENSAJE ESCRITO POR EL HIJO [%ld]\n”, (long) getpid());
     lon= strlen(buffer)+1;
     if ( write(fd, buffer, lon) != lon ) {
       fprintf(stderr, “Error al escribir en la fifo el hijo\n”);
       exit(1);
     }
     fprintf(stderr, “El hijo [%ld] termina\n”, (long) getpid());
     break;
```

Tuberías: Ejemplo

```
      default:  /* Código del proceso padre */
        fprintf(stderr, "Padre [%d] va a abrir la fifo %s\n", (long)getpid(),
                        argv[1]);
        if ( (fd=open(argv[1], O_RDONLY | O_NONBLOCK)) == -1) {
          fprintf(stderr, "El padre no puede abrir la fifo %s\n", argv[1]);
          exit(1);
        }
      fprintf(stderr, "Padre esperando para leer de la fifo... %s\n",
                       argv[1]);
      while ( (wait(&status)==-1) && (errno == EINTR) );
      if ( read(fd, buffer, TAM) <= 0) {
        fprintf(stderr, "El padre no puede leer de la fifo\n");
        exit(1);
      }
      fprintf(stderr, "Mensaje leído por el padre: %s\n", buffer);
  }
  exit(0);
}
```