Linux
Llamadas al Sistema

Manejo de archivos

- Para archivos regulares:
  - `fopen(), fwrite(), fprintf(), fscanf(), fread(), fseek(), fclose().`
  - Necesitan de un `FILE *pf` para referirse al archivo
  - `FILE` es una `struct`
- Para todo tipo de archivos:
  - `open(), write(), read(), close()`
  - Usan un descriptor de archivo `int fd`.

open()

- `int open(const char *pathname, int flags);`
- `pathname` es una cadena con el nombre del archivo
  - flags debe incluir alguno de los modos de acceso:
    - O_RDONLY, O_WRONLY, O_RDWR.
  - A esto se le adicionan Flags de Creación:
    - O_CLOEXEC, O_CREAT, O_DIRECTORY, O_EXCL, O_NOCTTY, O_NOFOLLOW, O_TMPFILE, O_TRUNC.
  - Flags de Estado:
    - O_APPEND, O_ASYNC, O_DSYNC, O_SYNC, O_NOATIME, O_LARGEFILE, O_NDELAY, O_PATH

otra forma de open()

- `int open(const char *pathname, int flags, mode_t mode);`
- A lo que se vio antes se le agrega mode
- es una combinación de flags que da los permisos de lectura, escritura y ejecución para el archivo
- Si no se usa se dan los permiso por defecto

Descriptores de archivo estándar

- En Linux hay tres descriptores de archivo pre-definidos, QUE SIEMPRE ESTAN ABIERTOS cuando se inicia un programa
  - stdin
  - stdout
  - stderr
- Para las funciones que necesitan un puntero tipo FILE se usan directamente:
- Para las funciones que necesitan un descriptor de archivo de tipo INT se usan las macros
  - STDIN_FILENO (0)
  - STDOUT_FILENO (1)
  - STDERR_FILENO (2)

Descriptores de archivo estándar

- `fprintf, fgets,fscanf` pueden usar `stdin, stdout y stderr` como indicación del archivo.
- `write y read usan las Macros`
- `Ejemplo:`
  - `fprintf(stderr, “El parametro no es correcto\n”);`
  - `fprintf(stdout, “Esto es un mensaje\n ”,);`
  - `fgets(stdin,buf,80);`
  - `read(STDIN_FILENO,buffer,5);`
  - `write(STDOUT_FILENO,buffer,11);`

PIPES

- Un pipe o tubería es un conducto unidireccional entre dos procesos que tiene un extremo de escritura y uno de lectura.
  - `#include <unistd.h>`
  - `int pipe(int pipefd[2]);`
  - `pipefd[0] extremo de lectura`
  - `pipefd[1] extremo de escritura`

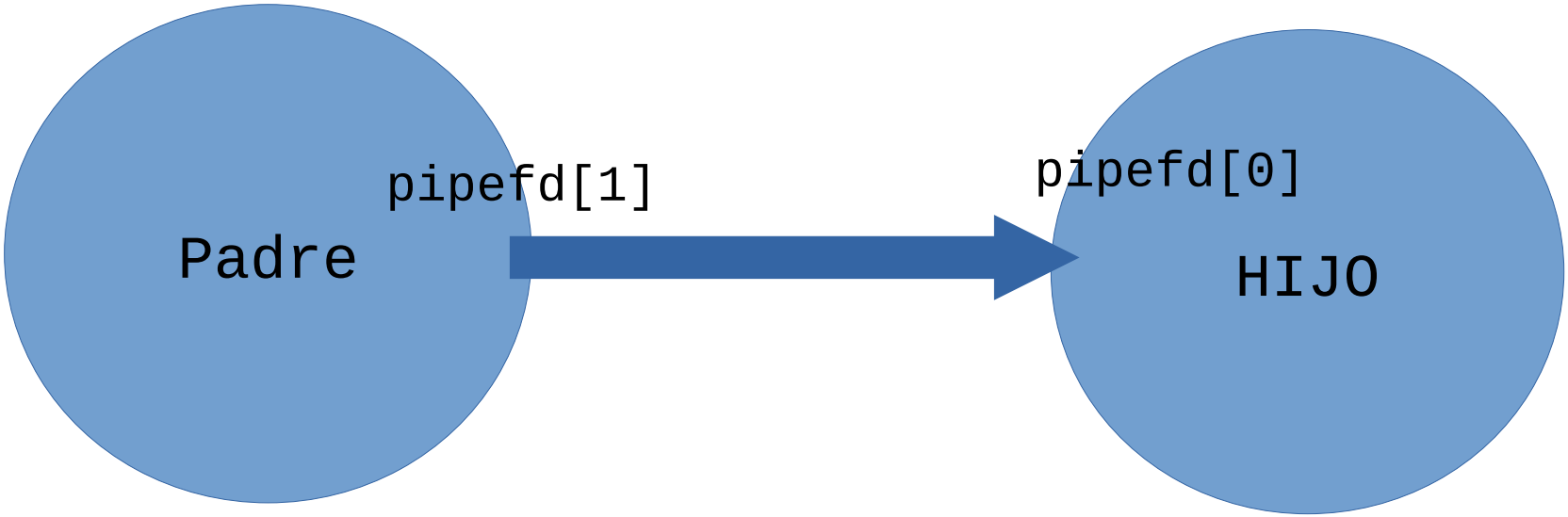
PIPES

- La manera mas sencilla de usarlos es crear un pipe en un proceso y luego hacer fork().
  - En ese caso el proceso hijo hereda los descriptores del pipe, y pueden cerrarse los extremos que no se usen.
  - close (pipefd[0]) cierro extremo de lectura
  - close (pipefd[1]) cierro extremo de escritura.
  - Si cierro uno en el hijo y el otro en el padre puedo obtener una comunicación unidireccional entre ambos procesos.
  - Si creo dos pipes puedo obtener comunicación bidireccional.

PIPES

creo pipefd[2]
cierro pipefd[0] porque
no voy a leer
Escribo en pipefd[1]

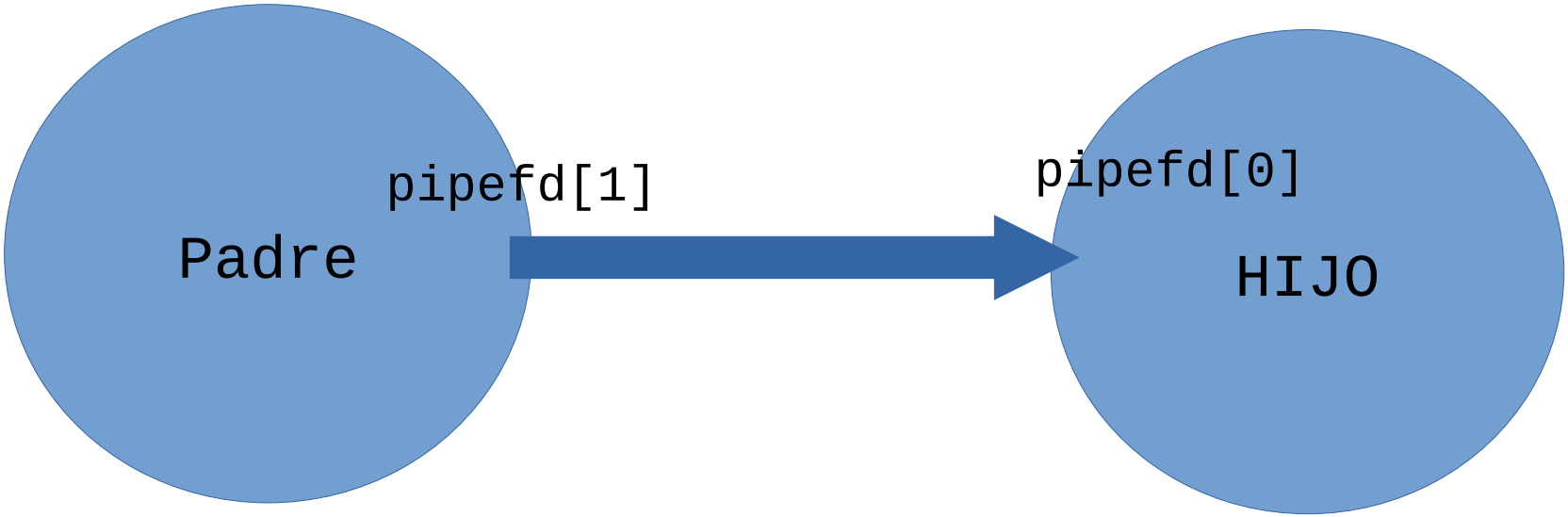
Heredo pipefd[2]
cierro pipefd[1] porque
no voy a escribir
Leo en pipefd[0]



PIPES

creo pipefd[2]
cierro pipefd[0] porque
no voy a leer
Escribo en pipefd[1]

Heredo pipefd[2]
cierro pipefd[1] porque
no voy a escribir
Leo en pipefd[0]



PIPES con nombre

```
#include <sys/types.h>
#include <sys/stat.h>
int mkfifo(const char *pathname, mode_t mode);
bash#>mkfifo nombre
Ej.
mkfifo pepe
ls>pepe
abrir otra consola . . .
cat<pepe
```