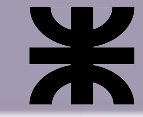

FIFO

FIFO – tuberías con nombre

- Una FIFO es similar a una tubería, con la diferencia de que una FIFO tiene un nombre en el sistema de archivo.
- Se abre de la misma manera que un archivo normal.
- Se pueden crear configurados como bloqueantes o no bloqueantes.
- Puede ser utilizado para la comunicación entre procesos no relacionados.
- Una vez que la FIFO se ha creado, cualquier proceso puede abrirlo.
- Utiliza las mismas funciones que utilizamos con tuberías y archivos: `read()`, `write()` y `close()`.
- Tiene un extremo de escritura y uno de lectura.
- Cuando **t**odos los descriptores de una FIFO se cierran, **los datos pendientes se descartan**, aunque la FIFO permanece.
- Los datos tienen persistencia de proceso.

Creación de una FIFO

La función para crear una FIFO es `mkfifo()`:

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

int mkfifo(nombre, mode_t mode);
```

Devuelve 0 si tuvo éxito o -1 en caso de error.

En el campo mode se colocan los permisos de la FIFO (lectura, escritura, ejecución) en formato octal o a través de una variable previamente definida (`#define VARIABLE octal`).

Apertura de una FIFO open()

La llamada al sistema `open()` crea un archivo nuevo o abre un archivo existente.

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

int open(nombre, int flags, ... /* mode_t mode */);
```

Devuelve el descriptor del archivo si tuvo éxito o -1 en caso de error.

El argumento `mode` indica los permisos al crear un archivo, en este caso este campo se coloca en cero. El argumento `flag` indica el modo de apertura:

1. `O_RDONLY` , abre el archivo sólo para lectura.
2. `O_WRONLY` , abre el archivo para sólo escritura.
3. `O_RDWR` , abre el archivo para lectura y escritura.
4. `O_NONBLOCK` , abre en modo no bloqueante.

Podemos abrir la FIFO de modo bloqueante o no bloqueante

- Una FIFO creada puede ser abierta por cualquier proceso (si tiene los permisos).
- La apertura de una FIFO para lectura (flag `O_RDONLY` en `open()`) bloquea al procesos hasta que otro proceso abra el FIFO para escritura (flag `O_WRONLY` en `open()`).
- De la misma forma, la apertura de la FIFO para escribir bloquea al proceso hasta que otro proceso abre la FIFO para lectura.
- Por lo dicho en el párrafo anterior la apertura de una FIFO sincroniza los procesos de lectura y escritura.
- Si queremos que el proceso no se bloquee debemos colocar el flag `O_NONBLOCK` en la llamada `open()`, o utilizar en la función `open()` el flag `O_RDWR` (esta opción no es la más recomendada).

Borrar una FIFO

```
#include <unistd.h>

int unlink(const char *pathname);
```

Devuelve 0 si tuvo éxito, o –1 en caso de error.

Crear una FIFO en línea de comando

Podemos crear una FIFO desde el shell con el comando `mkfifo`:

```
$ mkfifo --mode rwx pathname
```

`pathname` es el nombre de la FIFO. La opción `--mode` se utiliza para especificar los permisos de lectura, escritura y ejecución (`rwx`).

```
$ mkfifo --mode 777 ./my_fifo
```

Cuando listamos con el comando `ls` se muestra muestra la FIFO creada con un tipo `p` en la primera columna.

```
usuario@pc:~$ ls ./myfifo
prwxrwxrwx 1 usuario usuario 0 Apr 22 14:37 ./myfifo
```

Para borrar la FIFO desde un shell usamos el comando `rm`, como si fuera un archivo.

```
$ rm ./myfifo
```

Máscara de bits

Una máscara de bits junto a la operación lógica OR (`|`) se usa para poner en 1 determinados bits dentro de una variable. De esta forma, se indicará a una función (por ejemplo, `open()`) qué acción se debe tomar a partir del valor de la variable.

Las máscaras vistas O_RDONLY, O_WRONLY, O_RDWR, O_NONBLOCK están definidas en la biblioteca `fcntl.h` de la siguiente manera,

```
#define O_WRONLY 0b0000001
#define O_RDWR   0b0000010
```

En el siguiente ejemplo, ¿cuánto valdrá `or_result`?

```
int or_result;

or_result = (O_WRONLY | O_NONBLOCK);
fd = open(MY_FIFO , or_result, 0777 );
printf("or_result : %d\n", or_result);
```

Bibliografía

Kerrisk, Michael. *The linux programming Interface.* 2011. **Capítulo 44.**