

# Clase 4

# En una clase anterior

## Uso de archivos:

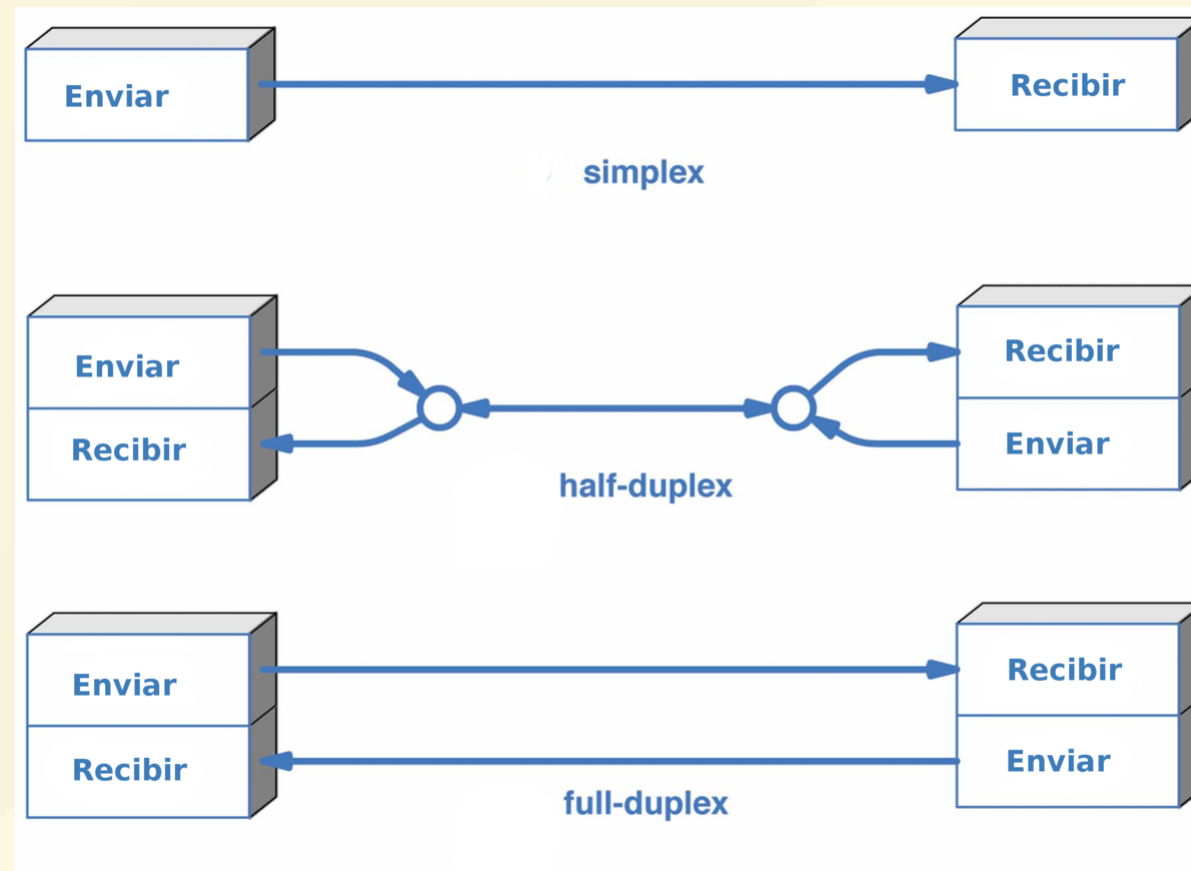
- Comunicación entre procesos no relacionados
  - Comunicación entre **varios** procesos
- Comunicación entre procesos vivos
- Capacidad de comunicar varios datos
- No son ni proveen un mecanismo de sincronización

# **Clase 4**

## **Pipes & Fifos**

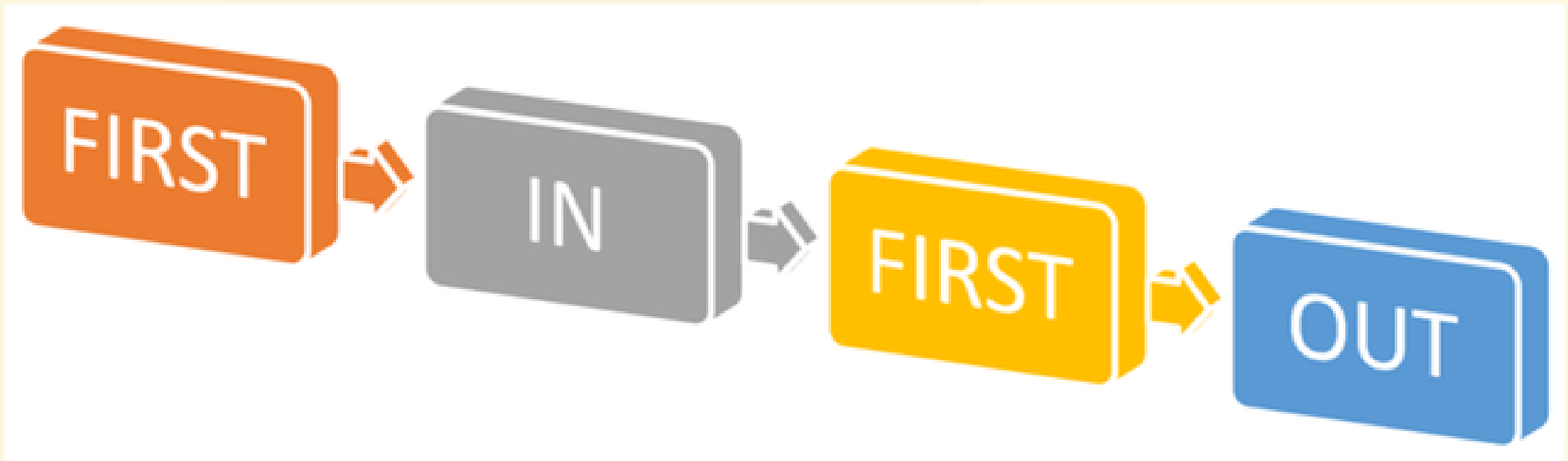
# Pipes & Fifos

Mecanismo de comunicación **unidireccional** entre procesos  
(*half-duplex*)



# Pipes & Fifos

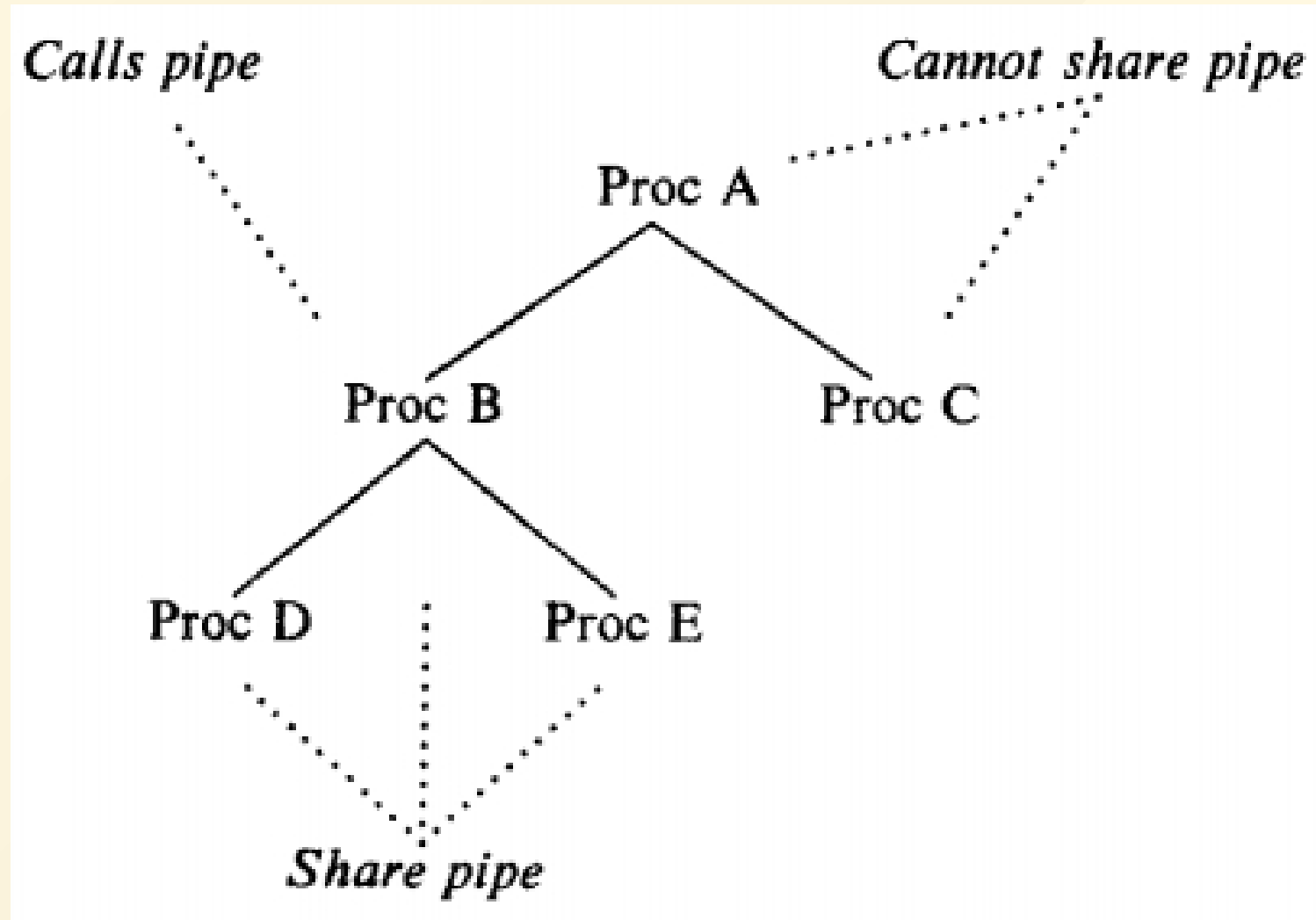
Se comportan como colas: First-In / First-Out



# Pipes & Fifos (Pipes)

- También conocidas como: *unnamed pipes*
- Los procesos que comunican deben tener relación entre si, ej:
  - Mismo padre
  - Ser padre e hijo

# Pipes & Fifos (Pipes)



# Pipes & Fifos (Fifos)

- También conocidas como: *named pipes*
- Los procesos que comunican pueden no tener relación alguna entre si
- Reside "físicamente" dentro del filesystem como archivo



# Operaciones

# Operaciones

Operación	Archivos	Pipes	Fifos
Crear	open	?	?
Abrir	open	?	?
Operar	read/write	?	?
Cerrar	close	?	?
Eliminar	unlink	?	?

# Creación

*Pipes*

pipe

# Creación (Pipes) `pipe`

```
int pipe ( int pipefd[2] );
```

- `pipefd`: Array de descriptores
  - Posición 0 - file descriptor de lectura
  - Posición 1 - file descriptor de escritura

⚠ Nota: Al ser un canal half-duplex, el proceso debe cerrar uno de los dos. **NO** se puede leer y escribir del mismo pipe

# Creación (Pipes) `pipe`

```
int pipe ( int pipefd[2] );
```

- Devuelve 0 en caso de éxito. Si falla, devuelve -1, y setea la variable externa `errno`

```
int file_descriptors [2];

int result = pipe(file_descriptors);

if (result == -1){
    perror("Error: ");
    exit(-1);
}

close(file_descriptors[1]);
```

# Creación

***FIFOs***

**mknod**

# Creación (FIFOs) `mknod`

```
int mknod ( const char* pathname, mode_t mode, dev_t dev );
```

- `pathname`: Nombre del archivo que representa el FIFO
- `mode`: `S_IFIFO` | permisos
- `dev`: Se ignora, poner `0`

Devuelve:

- 0 en caso de éxito.
- -1 en caso de error y setea la variable externa `errno`

# Operar



# Operaciones de lectura/escritura

Se puede operar sobre los pipes y fifos como si fueran archivos comunes, usando las funciones `read` y `write` sobre sus file descriptor:

- Lectura

```
ssize_t read (int fd, void* buf, size_t count);
```

- Escritura

```
ssize_t write (int fd, const void* buf, size_t count);
```

# Operaciones de lectura/escritura

Para conseguir el file descriptor de un FIFO, se debe abrir con open como si fuera un archivo común

- Lectura

```
int fd = open(path, O_WRONLY);
```

- Escritura

```
int fd = open(path, O_RDONLY);
```

# Operaciones de redirección

Existen funciones (de archivos), que permiten redireccionar la entrada y salida estándar un proceso

```
int dup(int oldfd);`
```

- Crea un file descriptor como copia del `oldfd` utilizando el menor descriptor posible
- Devuelve:
  - El file descriptor copiado en caso de éxito
  - -1 en caso de error y setea la variable externa `errno`

# Operaciones de redirección

```
int dup2(int oldfd, int newfd);`
```

- Copia el file descriptor `oldfd` en el descriptor `newfd`, cerrando primero `newfd` si es necesario
- Devuelve:
  - El file descriptor copiado en caso de éxito
  - -1 en caso de error y setea la variable externa `errno`

# Operaciones de redirección

## Reasignación de la entrada estándar

```
int file_descriptors [2];

int result = pipe(file_descriptors);
close(file_descriptors[1]);

dup2(file_descriptors[0],0);

// Leo un dato desde el pipe
char buffer[BUFSIZE];
std::cin.get ( buffer , BUFSIZE );
```

# Operaciones de redirección

## Reasignación de la salida estándar

```
int file_descriptors [2];

int result = pipe(file_descriptors);
close(file_descriptors[0]);

dup2(file_descriptors[1],1);

// Escribo un dato en el pipe
std::cout << "Hello pipe" << std::endl;
```

# **Destrucción**

# Destrucción

```
int close(int fd);
```

- Para eliminar un pipe, el proceso debe cerrar el file descriptor que dejó abierto
- Para eliminar un FIFO, el proceso debe cerrar el file descriptor que obtuvo con `open`

En el caso de los FIFOs, debe eliminarse también el archivo

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

- El archivo ya no se puede acceder por su nombre



# Recap

# Operaciones

Operación	Archivos	Pipes	FIFOs
Crear	<code>open</code>	<code>pipe</code>	<code>mknod</code>
Abrir	<code>open</code>	<code>pipe</code>	<code>open</code>
Operar	<code>read/write</code>	<code>read/write</code>	<code>read/write</code>
Cerrar	<code>close</code>	<code>close</code>	<code>close</code>
Eliminar	<code>unlink</code>	-	<code>unlink</code>

# **Sincronización y Comportamiento Bloqueante**

# Sincronización

- Por default, los pipes y FIFOs tienen un comportamiento bloqueante
  - Sirve como mecanismo de sincronización entre procesos
- Se puede utilizar en forma *no bloqueante*, pero **se desaconseja fuertemente su uso**
  - Pipes:

```
fcntl ( pipe_fd, F_SETFL, O_NONBLOCK );
```

- Fifos:
  - Utilizar `O_NONBLOCK` en `open`

**Bloqueante**  
**vs**  
**No Bloqueante**






Operación	Estado del pipe/FIFO	Retorno	
		Bloqueante	No Bloqueante
Abrir para lectura	Abierto para escritura	OK	OK
	No abierto para escritura	Se bloquea hasta que el FIFO sea abierto para escritura	OK
Abrir para escritura	Abierto para lectura	OK	OK
	No abierto para lectura	Se bloquea hasta que el FIFO sea abierto para lectura	OK

Operación	Estado del pipe/FIFO	Retorno	
		Bloqueante	No Bloqueante
Leer de un pipe/FIFO vacío	Abierto para escritura	Se bloquea hasta que haya datos o sea cerrado para escritura	Error EAGAIN
	No abierto para escritura	0 (EOF)	OK

Operación	Estado del pipe/FIFO	Retorno	
		Bloqueante	No Bloqueante
<b>Escribir en un pipe/FIFO</b>	Abierto para lectura	Si bytes ≤ PIPE_BUF es atómica si bytes > lugar_libre → se bloquea	Si bytes ≤ PIPE_BUF es atómica si bytes > lugar_libre → error EAGAIN
	No abierto para lectura	SIGPIPE (signal)	SIGPIPE (signal)



# Bibliografía

-  *The Design of the Unix Operating System*
  -  Maurice Bach
-  *Unix Network Programming, Interprocess Communications*
  -  W. Richard Stevens
-  Manuales del sistema operativo