Processes

Interprocesse communication: Pipes

Communication models

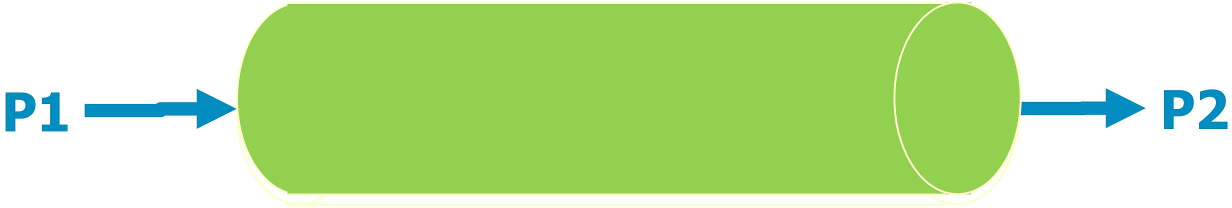
- The UNIX Interprocess Communication allows
  - **Half-duplex pipes**
  - FIFOs
  - Full-duplex pipes
  - Named full-duplex pipes
  - Message queues
  - **Semaphores**
  - Sockets
  - STREAMS
- Not all mechanisms are supported by all UNIX versions

Pipes

- Pipes are the oldest communication channel in UNIX systems
- A pipe is a data **data flow** between two processes
- Historically
  - The data flow in a pipe is **half-duplex**
    - Data flows only in one direction, from A to B or from B to A, but not in both directions (full-duplex)
  - Pipes can be used for processes communication with a **common parent**

Pipe's mode of operation

- Once the pipe is created, each process has acces to one end of the pipe through a file descriptor
- Since file descriptors have to be in common for the two processes, the involved processes must have a common ancestor
  - The pipe has to be created and then the fork can be performed (not the contrary)



System call pipe ( )

```
#include <unistd.h>

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

- A pipe can be created using the system call **pipe**

System call pipe ( )

```
#include <unistd.h>

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

- The function returns two file descriptors
- The vector **fileDescr** contains two new descriptors, so that:
  - fileDescr[0]: Opened for reading from the pipe
  - fileDesrc[1]: Opened for writing on the pipe
  - The output on fileDesc[1] corresponds to the input on fileDescr[0]

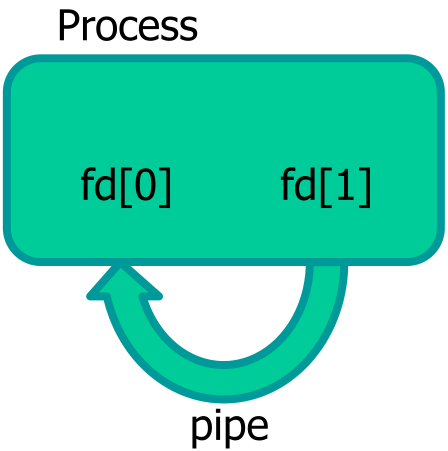
System call pipe ( )

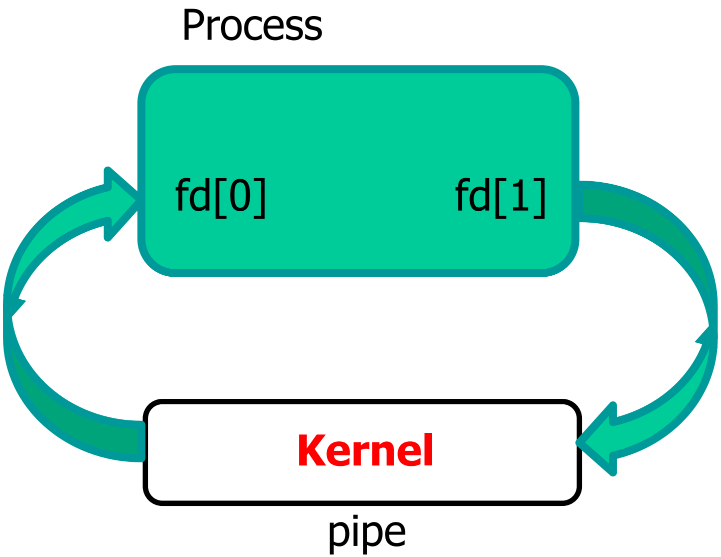
```
#include <unistd.h>

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

- Return value
  - The value is 0 if the operation succeded
  - The value is -1 if an error occurred

Pipe's use

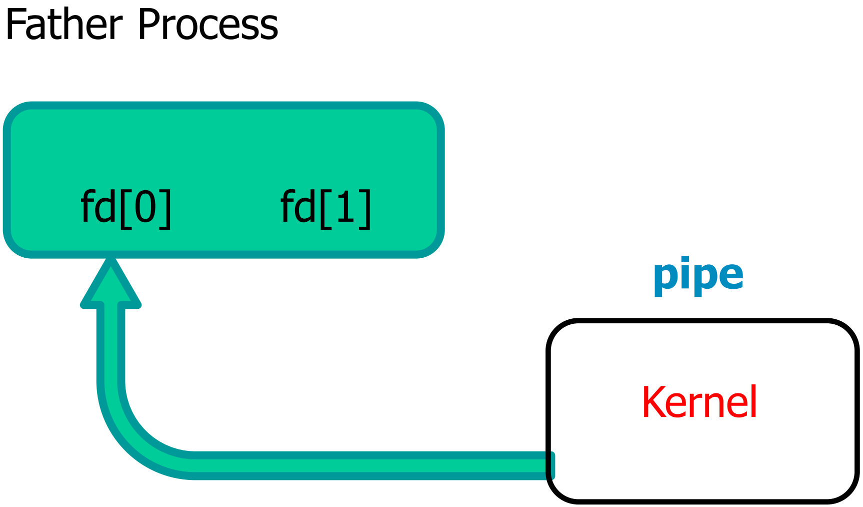
- A pipe inside the same process is nearly useless
  - The data flow through the pipe takes place through the kernel





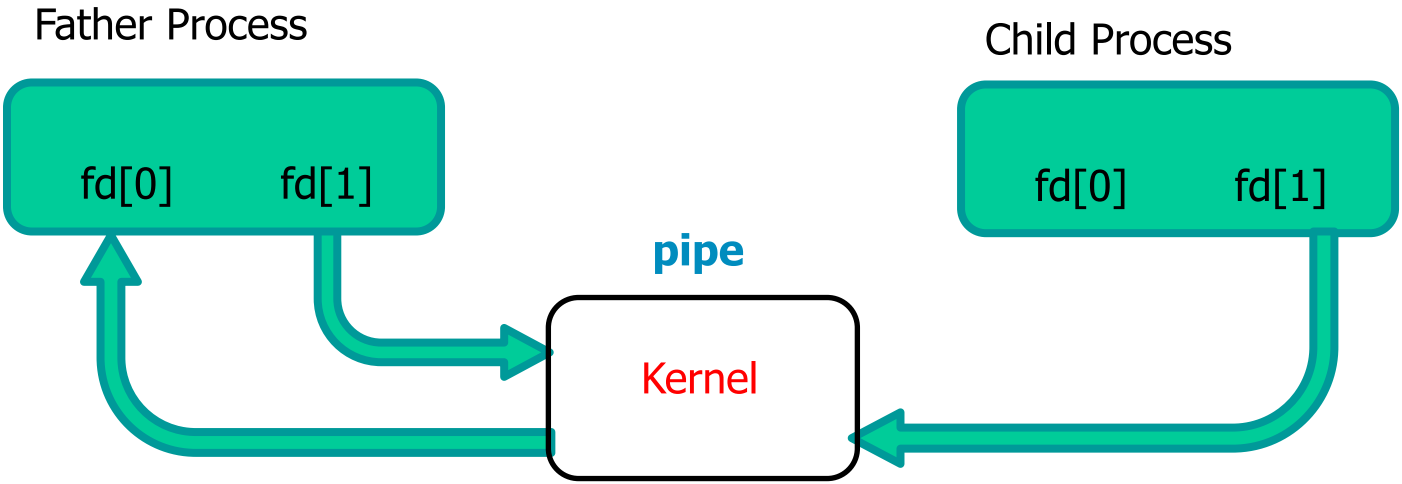
Pipe's use

- The standard flow of operation is:
  - A process creates a pipe



Pipe's use

- The standard flow of operation is:
  - A process creates a pipe; then is performs a fork
  - The child processes inherit the file descriptors



Father Process

fd[0] fd[1]

Child Process

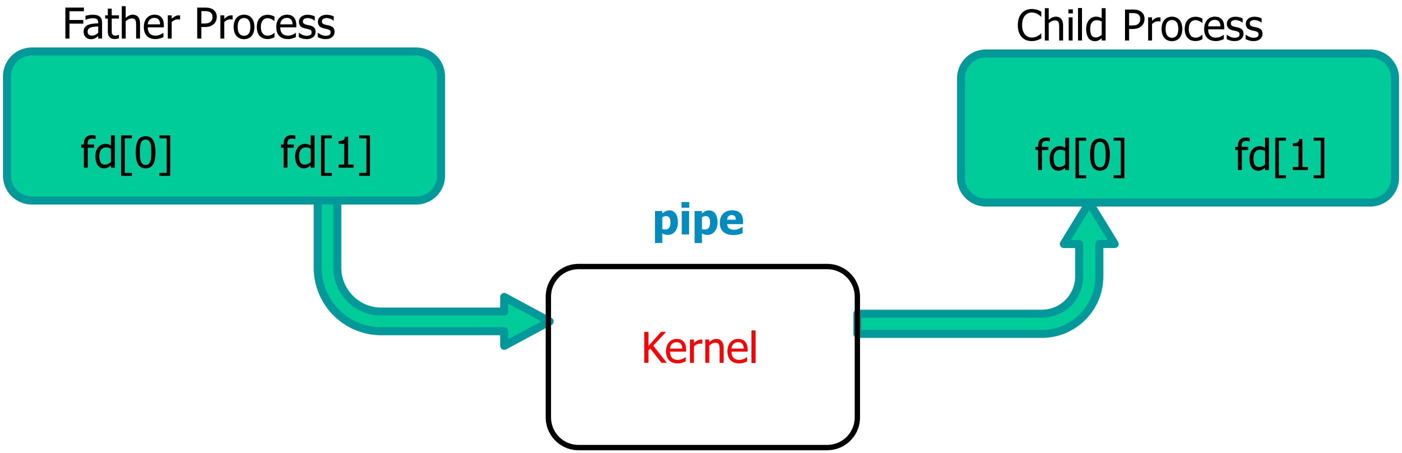
fd[0] fd[1]

pipe

Kernel

Pipe's use

- The standard flow of operation is:
  - A process creates a pipe; then it performs a fork
  - The child processes inherit the file descriptors
  - One of the two processes (e.g. father) writes in the pipe while the other (e.g. child) reads from the pipe
  - The unused descriptor can be closed



Pipe's I/O

- Reading from and writing on a pipe are performed through **read** and **write**
  - The pipe descriptor is an integer
  - The read system call
    - Reaturns only the available characters if the pipe contains less characters than what was asked
    - Blocks if the pipe is empty (**it's blocking**)
    - Returns 0 if the pipe the other end has been closed
  - The write system call
    - Blocks if the pipe is full (it's blocking)
    - Returns SIGPIPE if the other end has been closed

Example

- Create a pipe between a father process and a child process so that the two can exchange data
- Logic flow
  - Pipe creation
  - Process cloning
  - Closing the descriptor not used by each process
  - Read and write operation at pipe's ends

Example

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int main() {
  int n;
  int file[2];
  char cR = ‘X’;
  char cW;
  pid_t pid;
  if (pipe(file) == 0) {
    pid = fork ();
    if (pid == -1) {
      fprintf(stderr, "Fork failure");
      exit(EXIT_FAILURE);
    }
```

Example

```
  if (pid == 0) {
      // Child reads
      close (file[1]);
      n = read (file[0], &cR, 1);
      printf("Read %d bytes: %c\n", n, cR);
      exit(EXIT_SUCCESS);
    } else {
      // Parent writes
      close (file[0]);
      n = write (file[1], &cW, 1);
      printf ("Wrote %d bytes: %c\n", n, cW);
    }
  }
  exit(EXIT_SUCCESS);
}
```