Operating Systems

(Sistemas Operativos)

Guide 4: Pipes

University of Minho

2024 - 2025

Process API

Inter-Process Communication

Through regular files

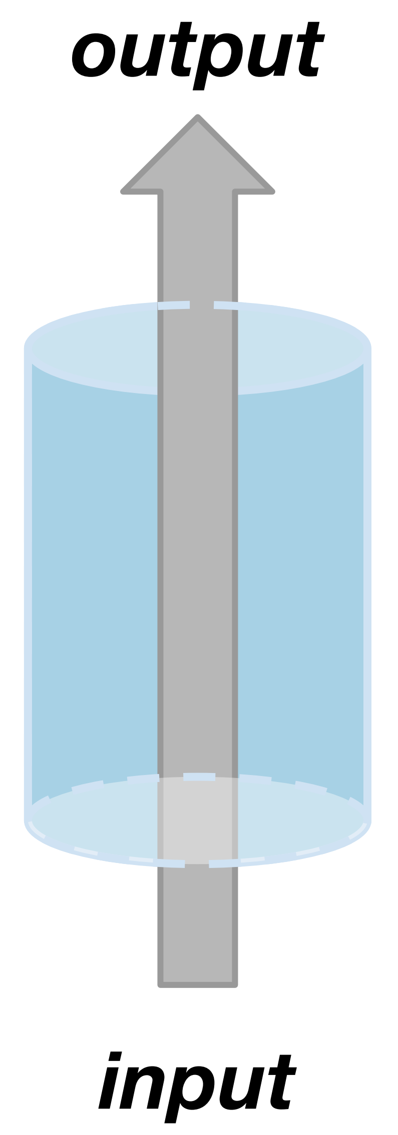
- file writing and reading **must be carefully synchronized**
- data is written to disk (**slow**)
- requires management of names, permissions and **inter-process interference**

Process API

Inter-Process Communication

Anonymous Pipes

- **Between related processes** (e.g., parent - child, children of the same parent).
- Produced (written) data is **kept in a memory region** to be consumed (read).
- The kernel handles writers (producers) and readers (consumers).
- **Writers block (wait) if there is no available space, and readers block if there is no data.**
- Data flows in a **one-way First-in First-out** (**FIFO**) manner.
- Enables chaining of programs without modifying them (with the help of the other system calls).
  - E.g., *$ ls* **|** *less*

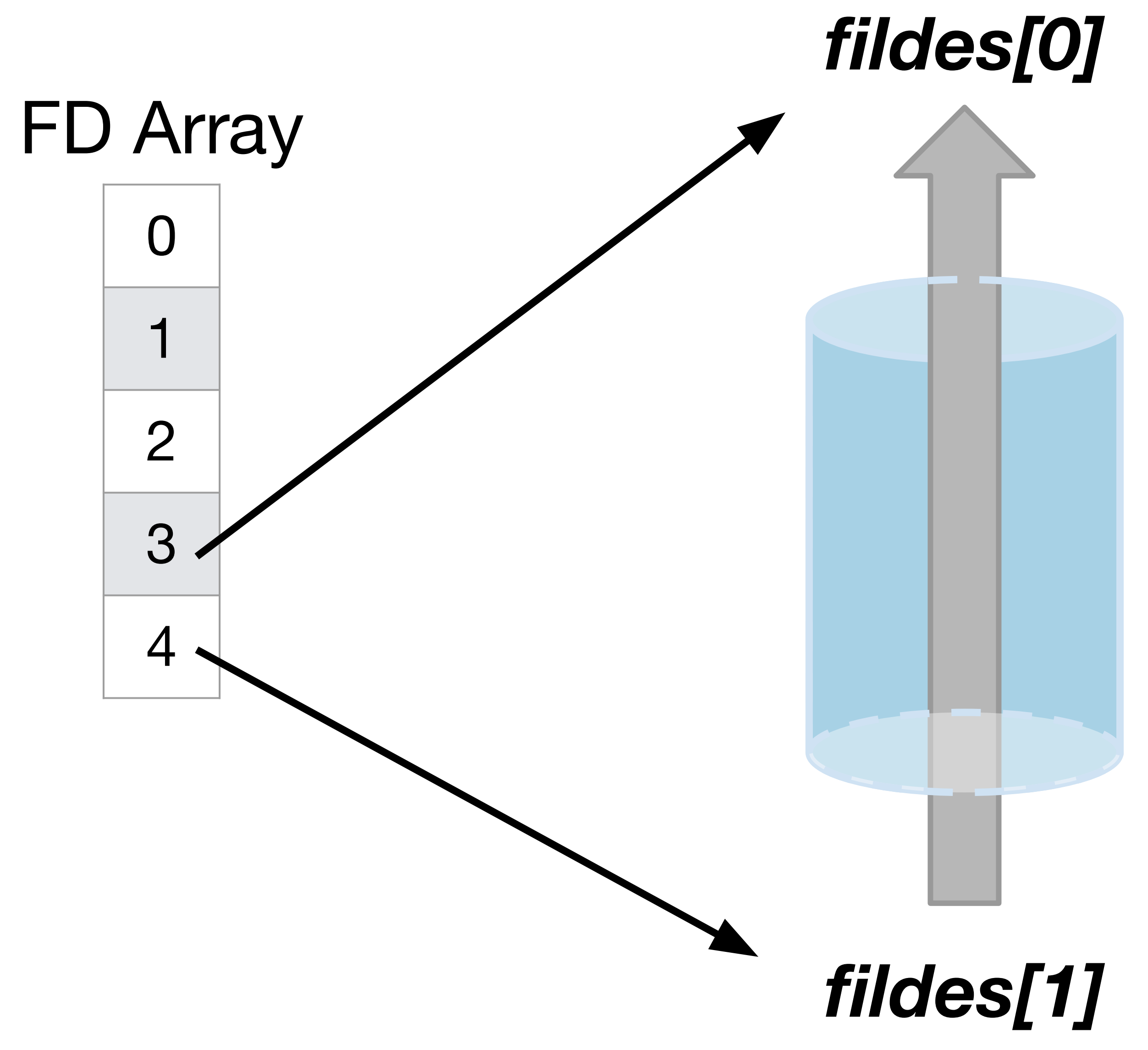


Process API

Anonymous Pipes

#include <unistd.h>

- *int **pipe**(int fildes[2])*
  - **fildes:** array populated by the function with the *FDs* of the write and read ends of the pipe.
  - Returns: 0 on success, -1 otherwise



Considerations:

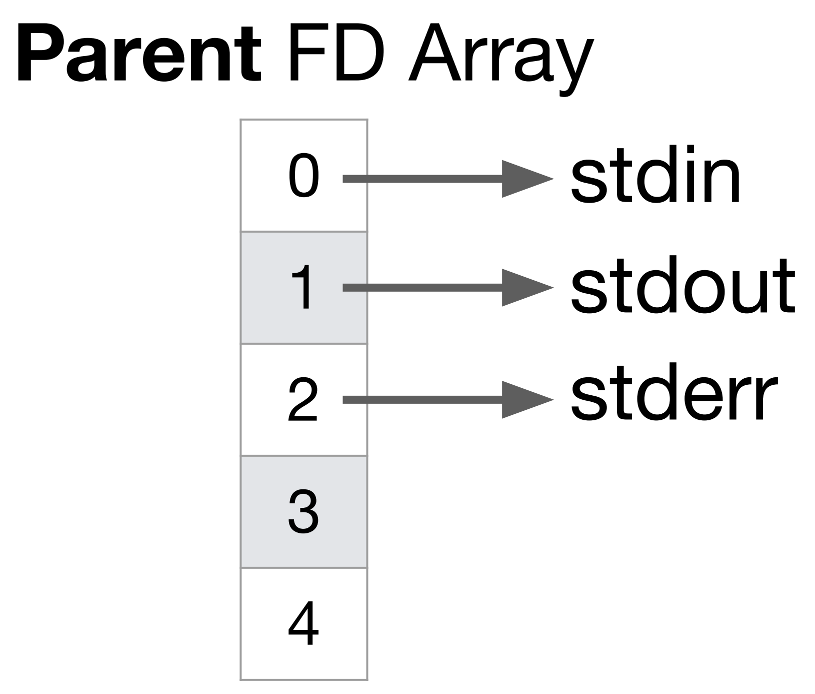
1. **Data written** to `fildes[1]` (write end) **can be read** from `fildes[0]` (read end).
2. Reading from `fildes[0]` reaches **EOF only when `fildes[1]` is closed.**
3. **Processes that read from the pipe should close the write end, and vice-versa**
4. **Leaving unnecessary pipe ends open can lead to deadlocks.**
5. **Writing to a pipe whose read end is closed results in the process being terminated.**

For more information: *$ man 2 pipe*

Process API

Example: Anon Pipe + Fork

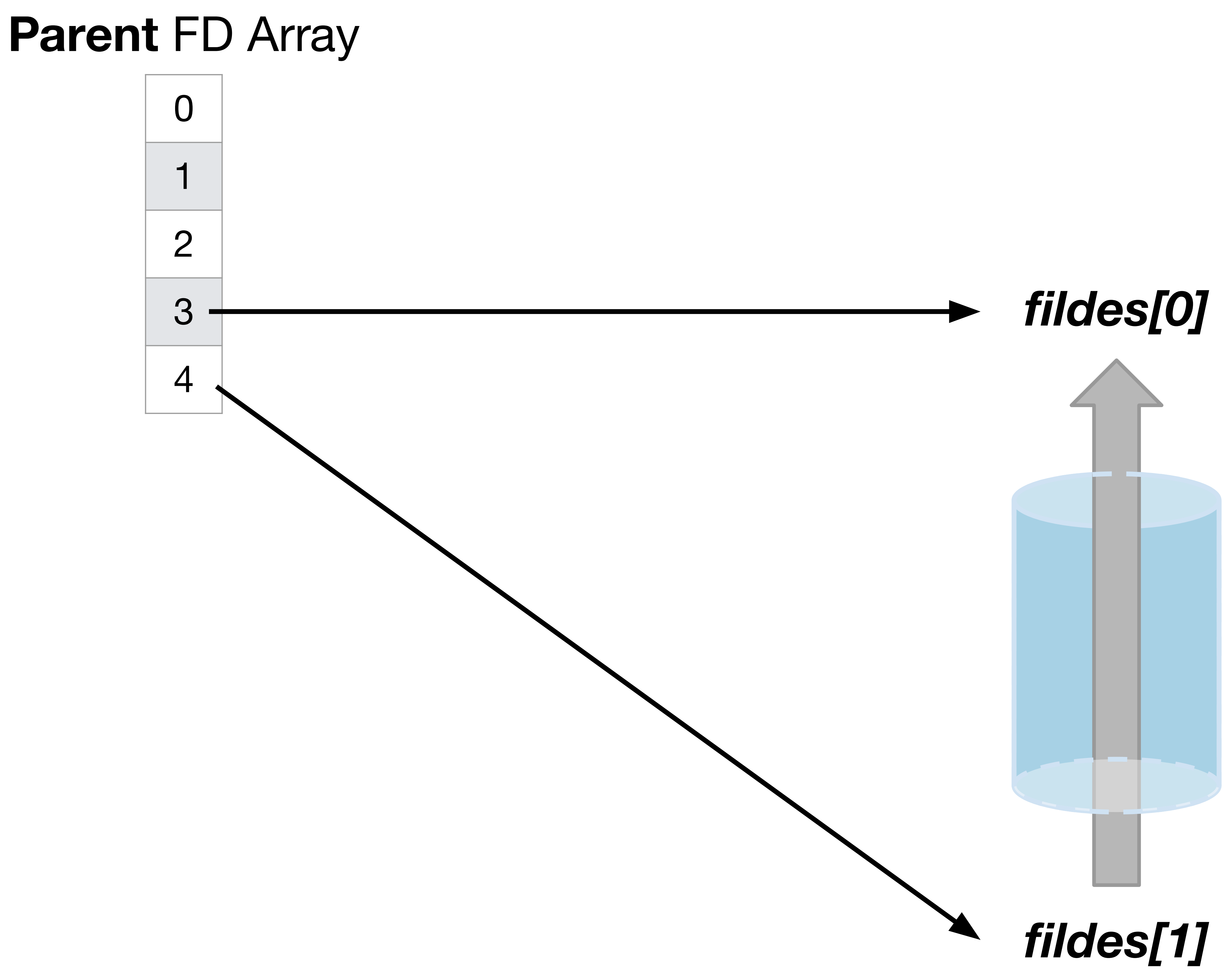
```c
int main() {
  pid_t pid;
  int fildes[2];
  pipe(fildes);
  if ((pid = fork()) == 0) {
    close(fildes[0])
    // child process
  } else {
    close(fildes[1])
    // parent process
  }
  return 0;
}
```



Process API

Example: Anon Pipe + Fork

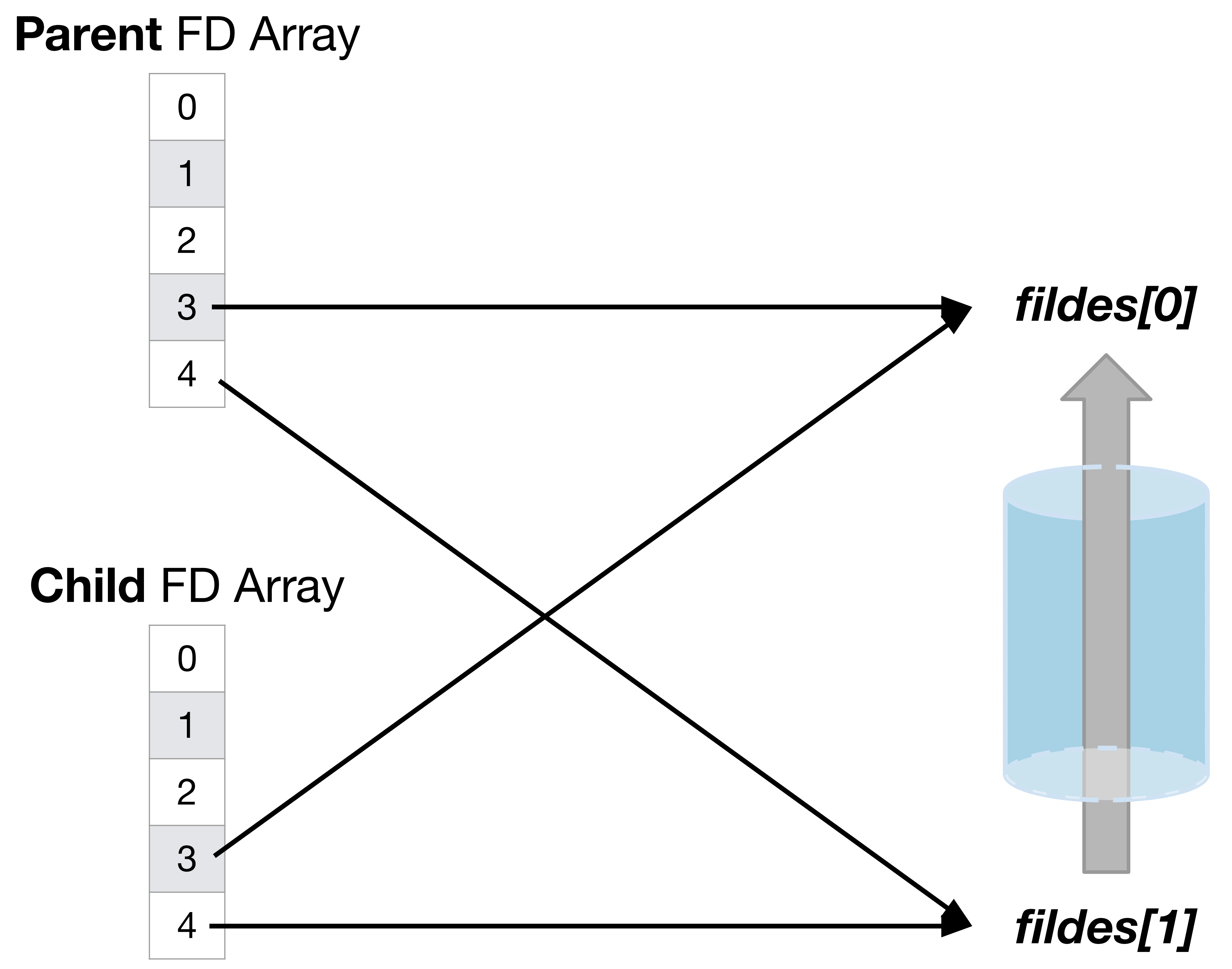
```c
int main() {
  pid_t pid;
  int fildes[2];
  pipe(fildes);
  if ((pid = fork()) == 0) {
    close(fildes[0])
    // child process
  } else {
    close(fildes[1])
    // parent process
  }
  return 0;
}
```



Process API

Example: Anon Pipe + Fork

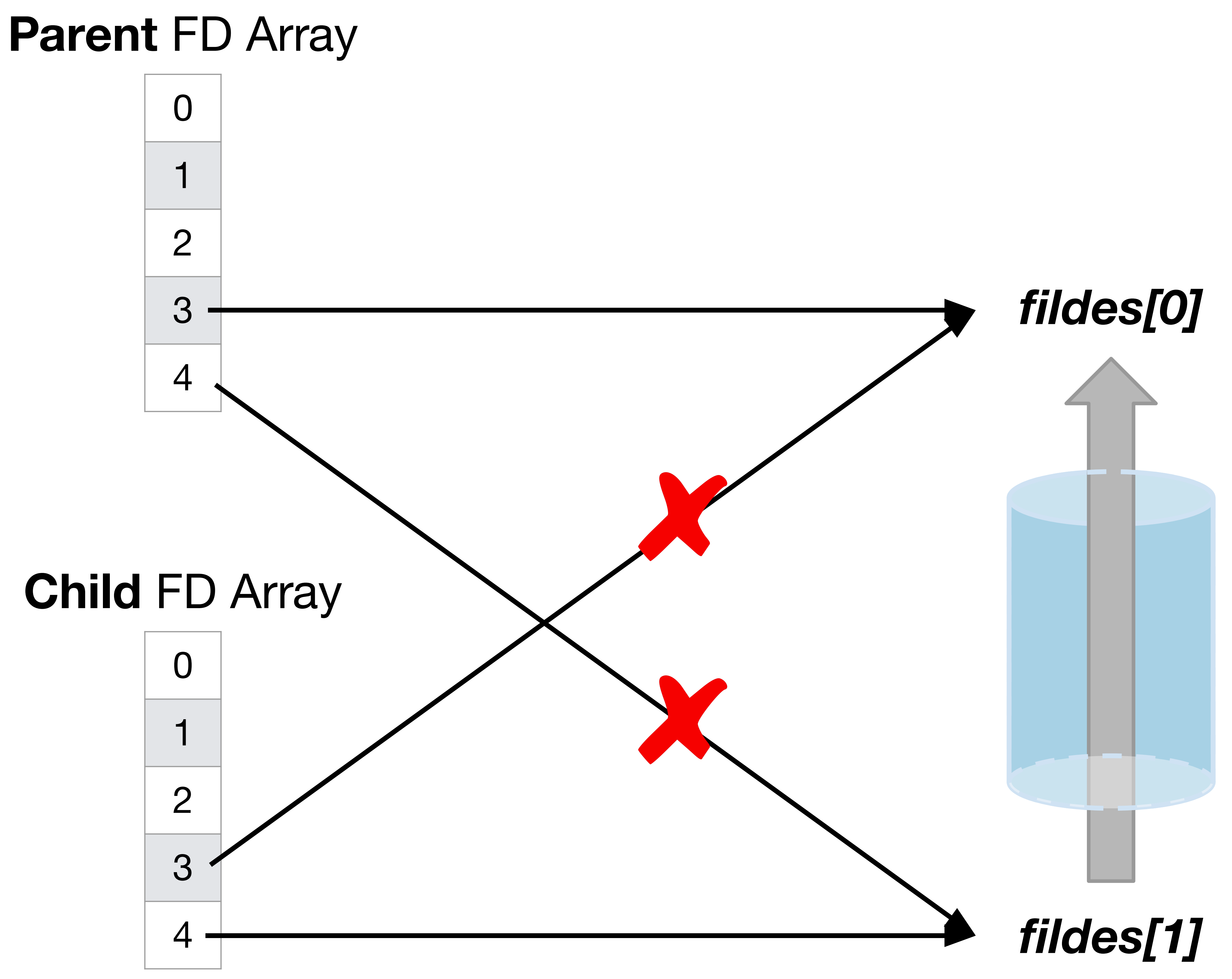
```c
int main() {
  pid_t pid;
  int fildes[2];
  pipe(fildes);
  if ((pid = fork()) == 0) {
    close(fildes[0])
    // child process
  } else {
    close(fildes[1])
    // parent process
  }
  return 0;
}
```



Process API

Example: Anon Pipe + Fork

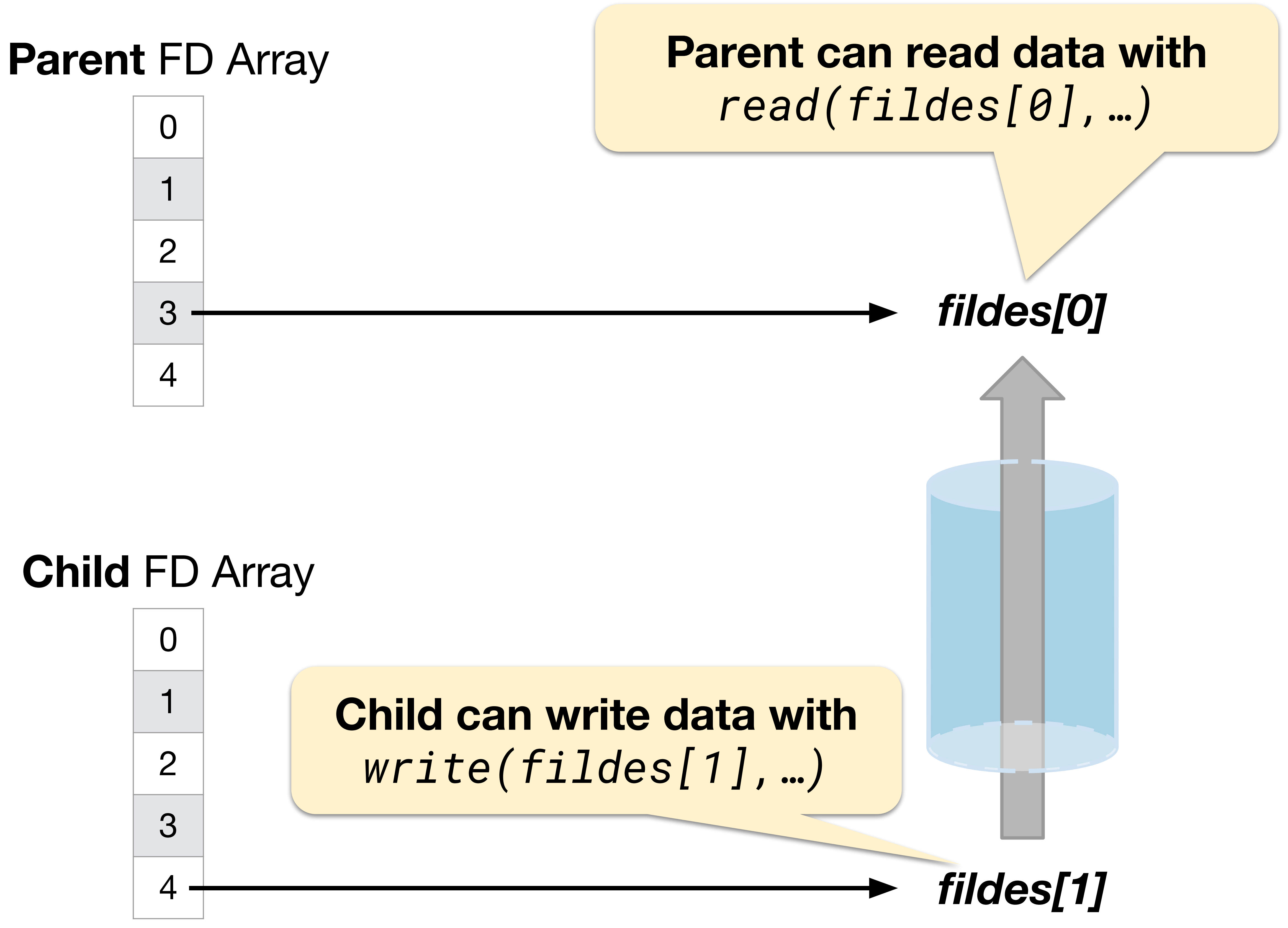
```c
int main() {
  pid_t pid;
  int fildes[2];
  pipe(fildes);
  if ((pid = fork()) == 0) {
    close(fildes[0])
    // child process
  } else {
    close(fildes[1])
    // parent process
  }
  return 0;
}
```



Process API

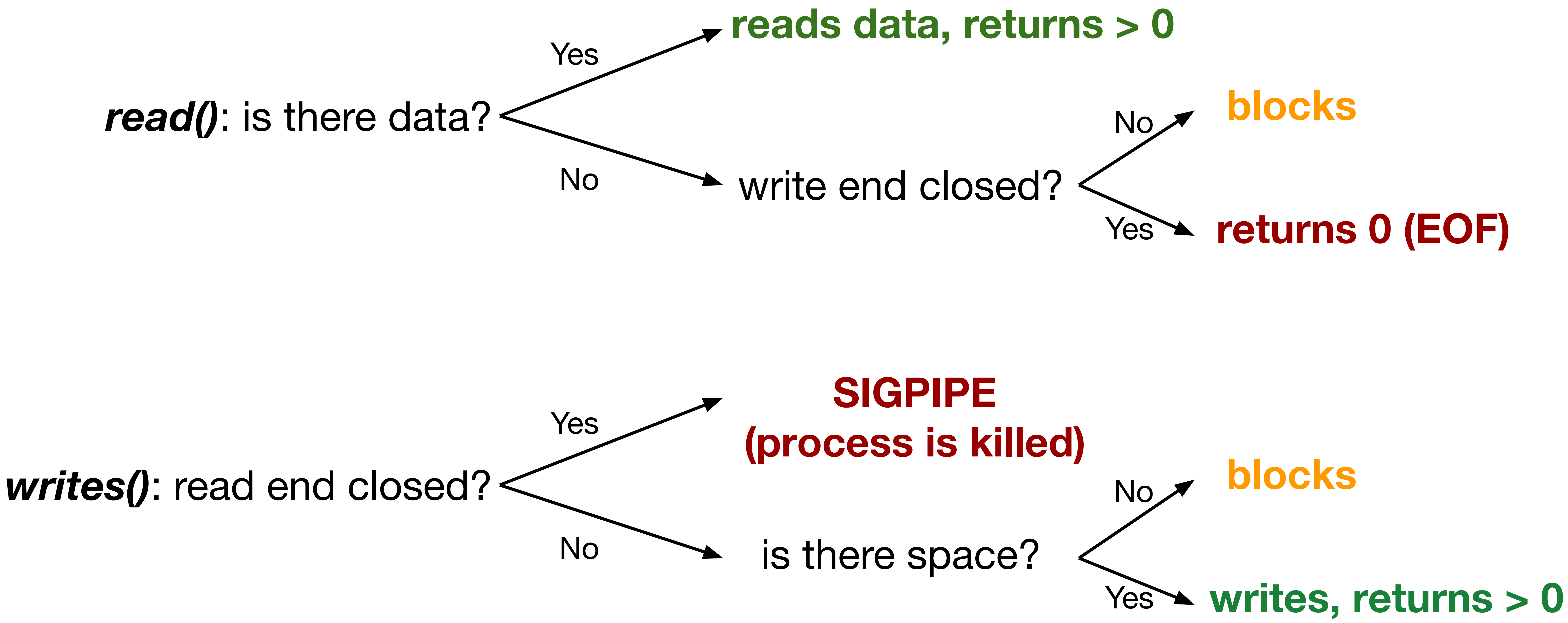
Example: Anon Pipe + Fork

```c
int main() {
  pid_t pid;
  int fildes[2];
  pipe(fildes);
  if ((pid = fork()) == 0) {
    close(fildes[0])
    // child process
  } else {
    close(fildes[1])
    // parent process
  }
  return 0;
}
```


Parent FD Array
0
1
2
3
4
Parent can read data with *read(fildes[0], …)*
fildes[0]
Child FD Array
0
1
2
3
4
Child can write data with *write(fildes[1], …)*
fildes[1]

Process API

Anonymous Pipes


read(): is there data?
Yes
reads data, returns > 0
No
write end closed?
No
blocks
Yes
returns 0 (EOF)
writes(): read end closed?
Yes
SIGPIPE
(process is killed)
No
is there space?
No
blocks
Yes
writes, returns > 0