Input / output

Computadors

Grau en Ciència i Enginyeria de Dades

Xavier Verdú, Xavier Martorell

Facultat d'Informàtica de Barcelona (FIB)

Universitat Politècnica de Catalunya (UPC)

2019-2020 Q2

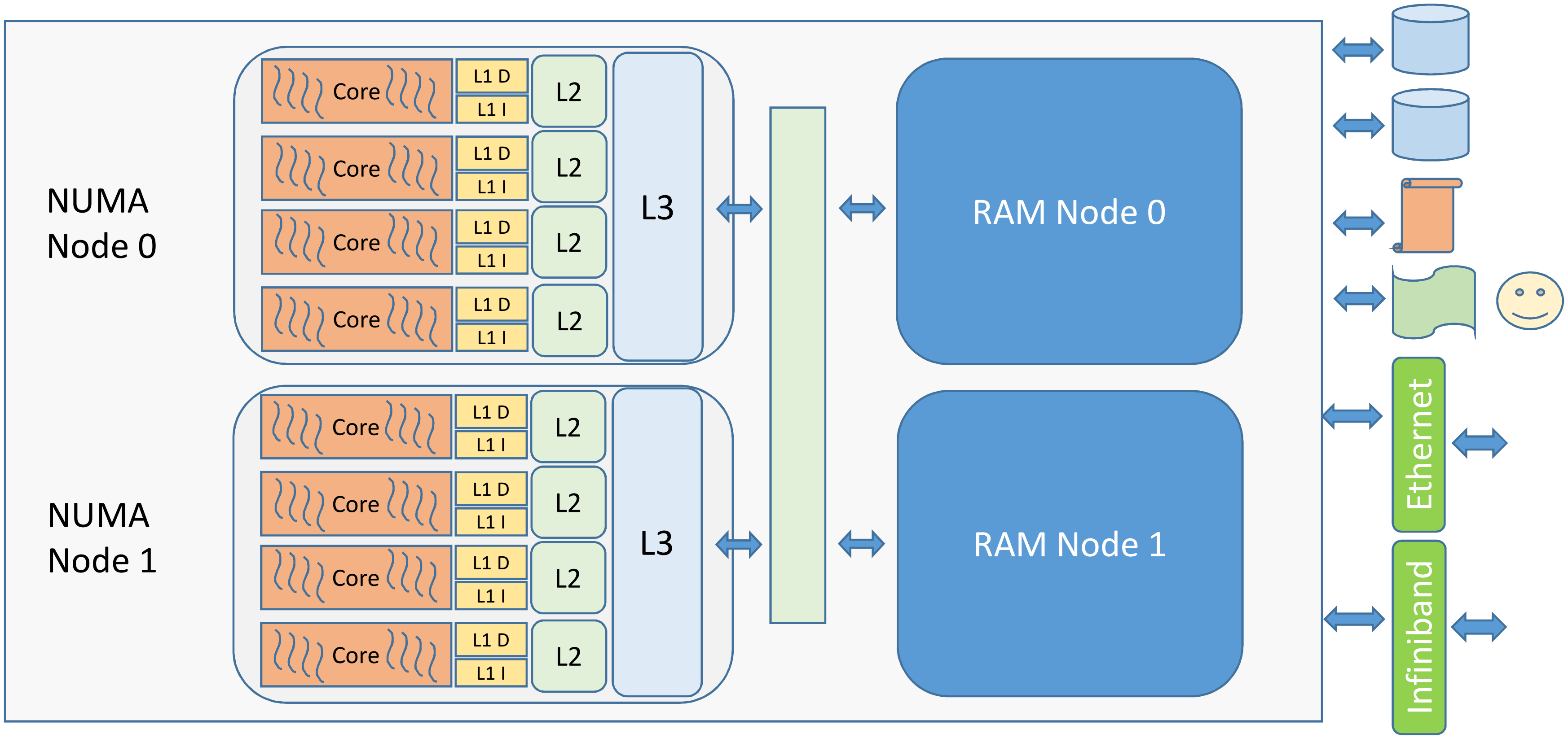
Creative Commons License

Table of Contents

Introduction

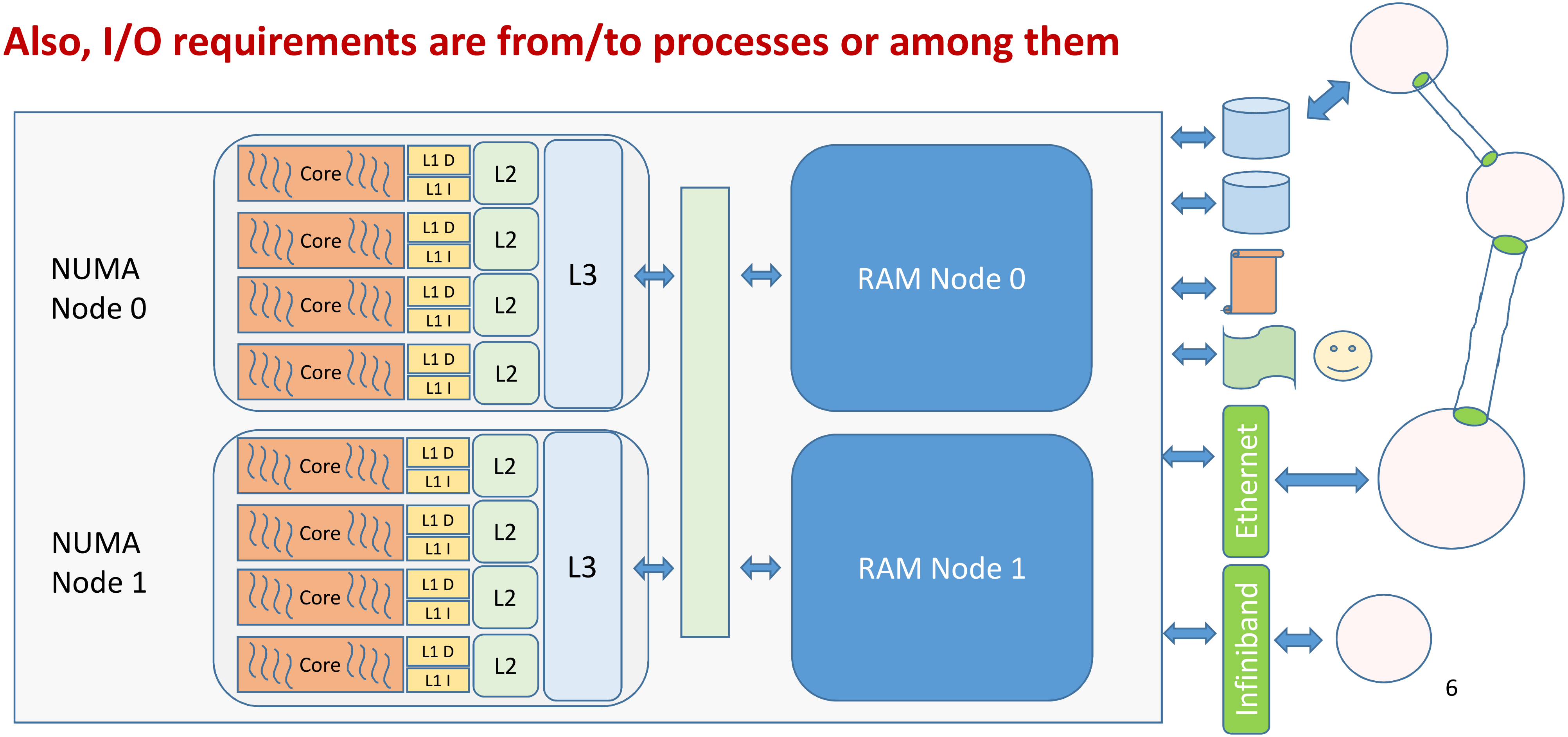
Introduction

- Need to move data in to/out of the main system, onto permanent storage or other devices



Introduction

- **Also, I/O requirements are from/to processes or among them**

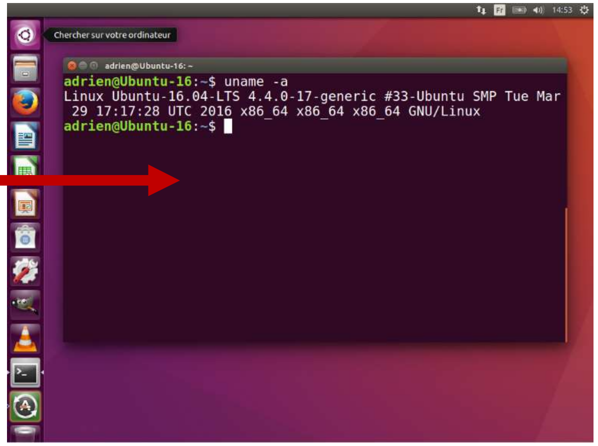


Introduction

- I/O is one of the key activities of the processes
  - Always from the point of view of the process: Input / Output
  - It is also used for process communication: pipes, internet sockets
- Access to devices is a privileged operation
  - It needs to be secure to have an usable, multi-user environment
  - Root / administrator
- Once accessed for the first time (open, pipe, socket...)
  - Access is inherited through the parent/child process relationship
- Threads on a process share access to files and devices

OS support for I/O subsystem

- Any I/O device needs an addressable representation
  - In UNIX/Linux devices are represented in the File System
    - E.g.: a terminal: "/dev/pts/12"
  - More details on File Systems on the next lesson



- OS introduces multiple abstraction layers
  - To simplify I/O management
- Every PCB – process – has a private table of "virtual" devices
  - **File descriptor table**
    - By default, three entries initialized: 0→STDIN; 1→STDOUT; 2→STDERR
  - Entries point to other global internal data structures of the kernel
    - Open files table

Device Independence

- The enormous variety of devices needs to be managed in some way
  - Categories according to: speed, device type, transfer type, etc.

- Internal classification of devices
  - UNIX/Linux: **character/block**,
    major (what device type is?)
    minor (what instance is?)

- Main Idea: Device Independence Basis
  - **Uniform I/O operations** – open, close, read, write...
  - **Virtual devices** – pseudoterminals, disk partitions...
  - **Redirection** – implementation of >f >>f >&f </dev/null ... | ...

Device types

- Character devices - Access can be done byte to byte
  - Terminals, consoles, serial lines, keyboard, mouse, screen...
  - Printers
  - Real time clock (rtc)
  - Basic data management & testing
    - null, zero, full, random, urandom
  - Kernel
    - Messages, memory, I/O ports, physical memory
  - Hardware
    - Firmware, hardware registers

Device types

- Character devices - Access can be done byte to byte

```
crw-r-----  1 root kmem      1,   1 Jan 16 08:14 mem
crw-r-----  1 root kmem      1,   2 Jan 16 08:14 kmem
crw-rw-rw-  1 root root      1,   3 Jan 16 08:14 null
crw-r-----  1 root kmem      1,   4 Jan 16 08:14 port
crw-rw-rw-  1 root root      1,   5 Jan 16 08:14 zero
crw-rw-rw-  1 root root      1,   7 Jan 16 08:14 full
crw-rw-rw-  1 root root      1,   8 Jan 16 08:14 random
crw-rw-rw-  1 root root      1,   9 Jan 16 08:14 urandom
crw-r--r--  1 root root      1,  11 Jan 16 08:14 kmsg
```

Device types

- Block devices – Access is done on a block by block basis
  - Disks /dev/sda, /dev/sdb...
    - SCSI, SATA, ATA
    - IDE (/dev/hda, /dev/hdb...)
  - CD/DVD /dev/dvdrom
  - RAMDisks /dev/ram0
  - Loopback device – Disk device on a file
    - /dev/loop0

Device types

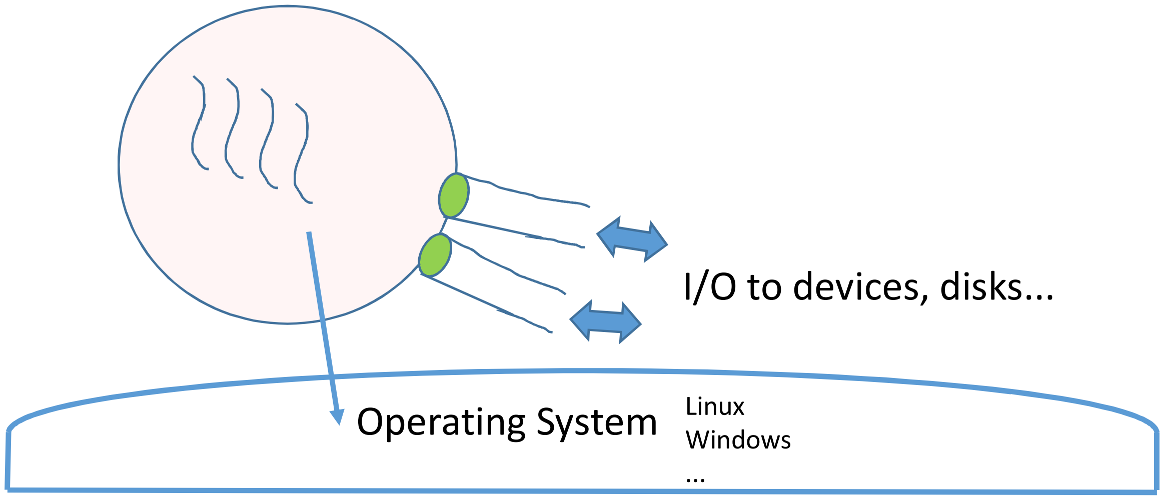
- Block devices – Access is done on a block by block basis

```
brw-rw----  1 root disk      8,   0 Jan 16 08:14 sda
brw-rw----  1 root disk      8,   1 Jan 16 08:14 sda1
brw-rw----  1 root disk      8,   2 Jan 16 08:14 sda2

brw-rw----  1 root disk      7,   0 May 28 10:46 loop0
brw-rw----  1 root disk      7,   1 Jan 16 08:14 loop1
brw-rw----  1 root disk      7,   2 Jan 16 08:14 loop2
brw-rw----  1 root disk      7,   3 Jan 16 08:14 loop3
brw-rw----  1 root disk      7,   4 Jan 16 08:14 loop4
brw-rw----  1 root disk      7,   5 Jan 16 08:14 loop5
brw-rw----  1 root disk      7,   6 Jan 16 08:14 loop6
brw-rw----  1 root disk      7,   7 Jan 16 08:14 loop7
```

The file descriptor table

- Each process has a file descriptor table
  - It is the gateway to manage the devices totally independent of characteristics
  - On each process, file descriptors are numbered 0, 1, 2, ...
  - When allocating (opening) a new file descriptor, the lowest available number is used

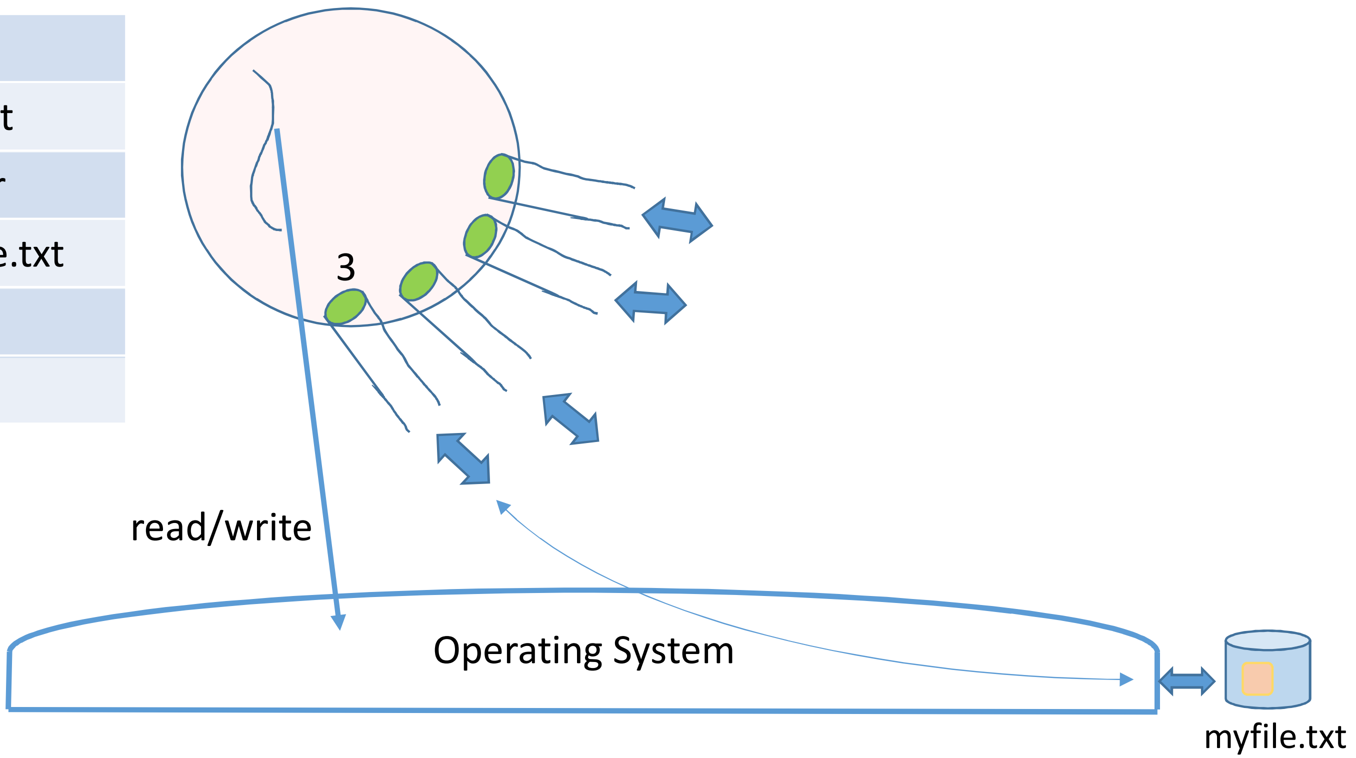


The file descriptor table

- Processes and file descriptors

In PCB

0	stdin
1	stdout
2	stderr
3	myfile.txt
4	
…	…



File descriptor table inheritance

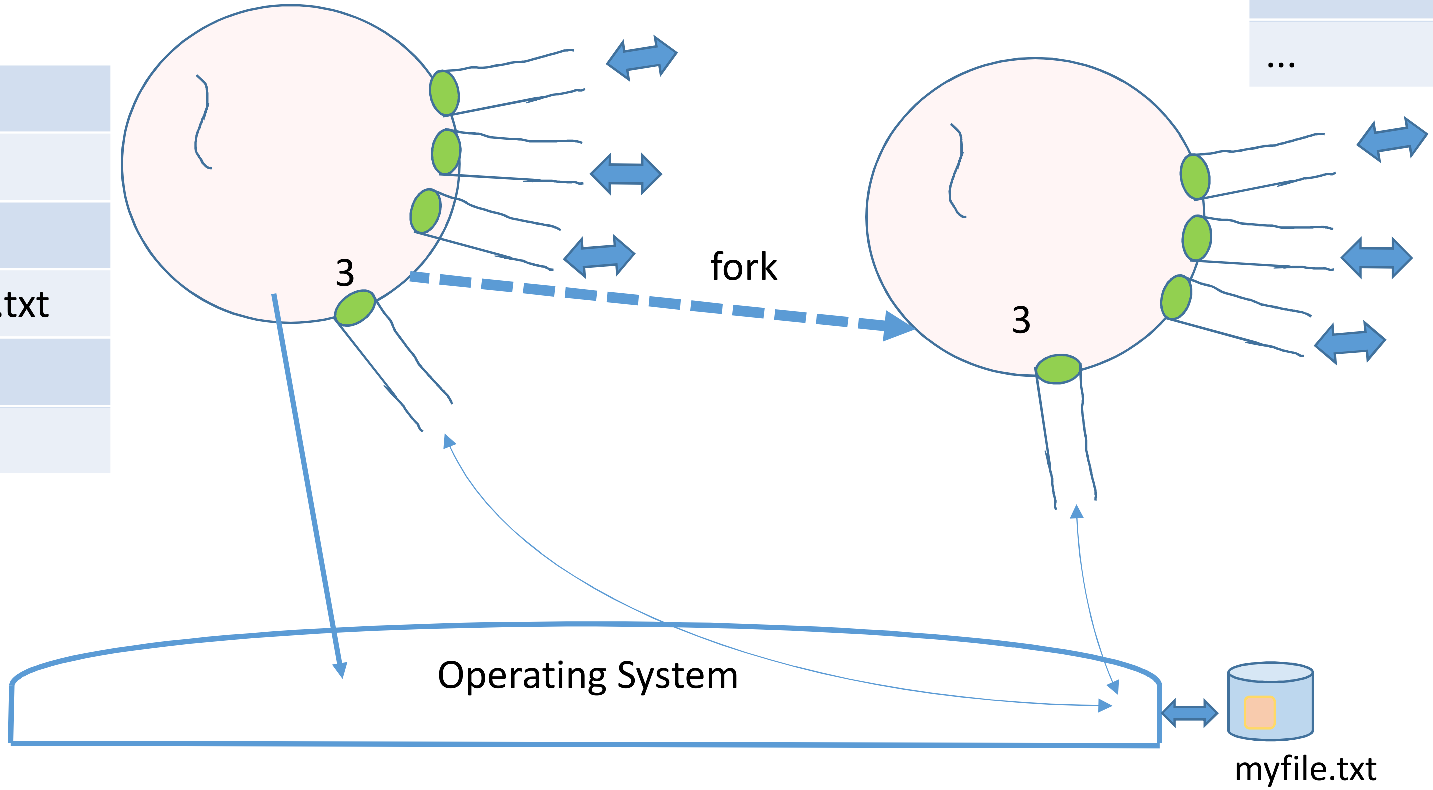
- Inheritance of open file descriptors on fork

In PCB1

0	stdin
1	stdout
2	stderr
3	myfile.txt
4	
…	…

In PCB2

0	stdin
1	stdout
2	stderr
3	myfile.txt
4	
…	…



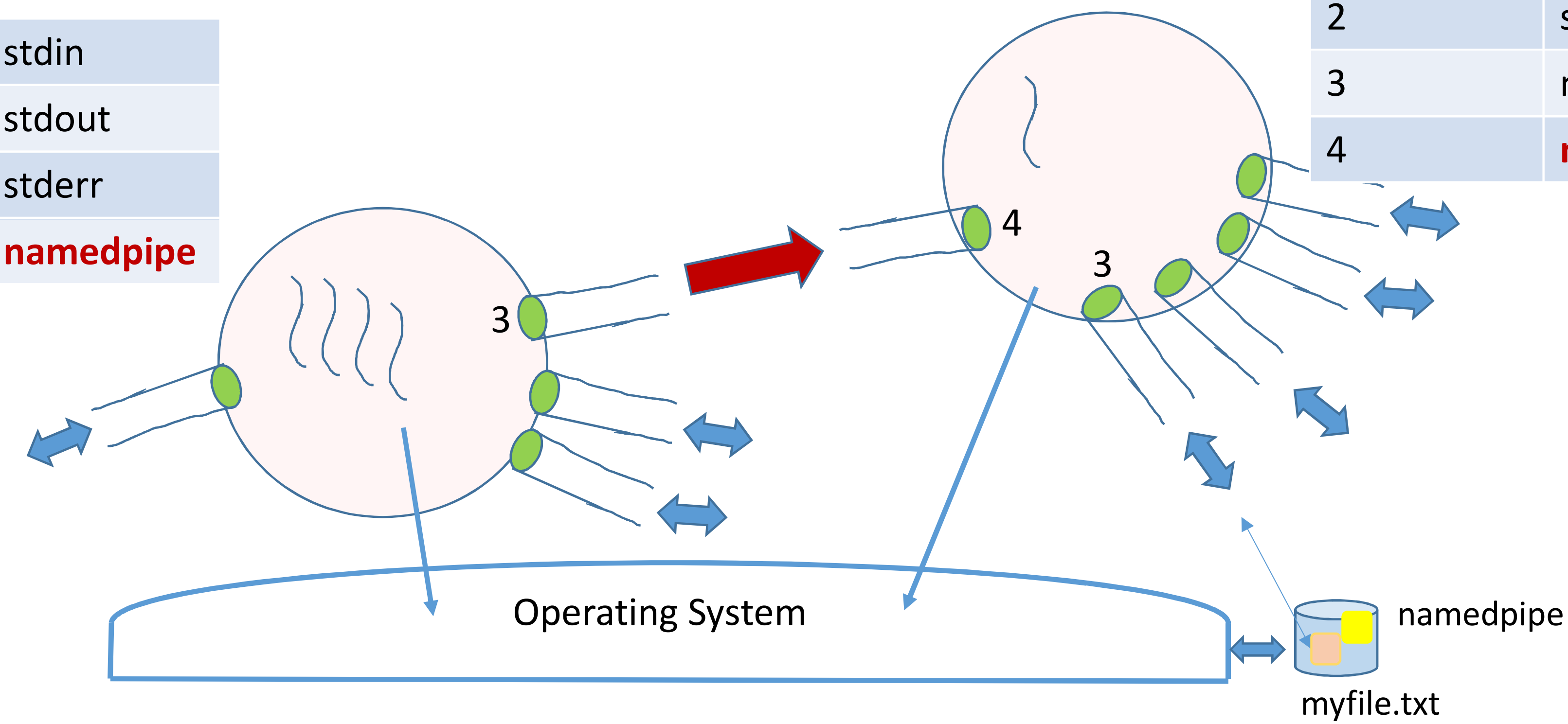
Process communication: named pipe

- open (“namedpipe”, O_WRONLY)

open(“namedpipe”, O_RDONLY)

0	stdin
1	stdout
2	stderr
3	**namedpipe**

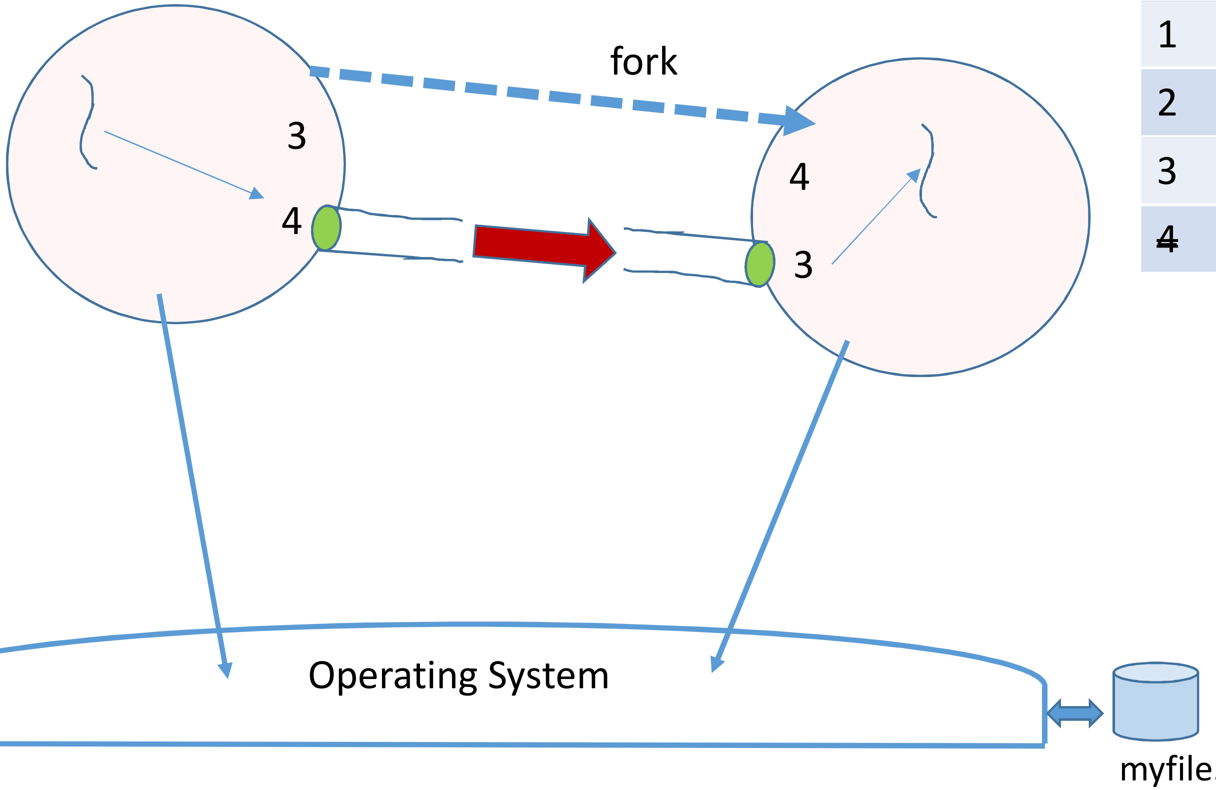
0	stdin
1	stdout
2	stderr
3	myfile.txt
4	**namedpipe**



Process communication: unnamed pipe

- pipe (fd) and sharing with the child

0	stdin
1	stdout
2	stderr
~~3~~	~~fd[0]~~
4	fd[1]



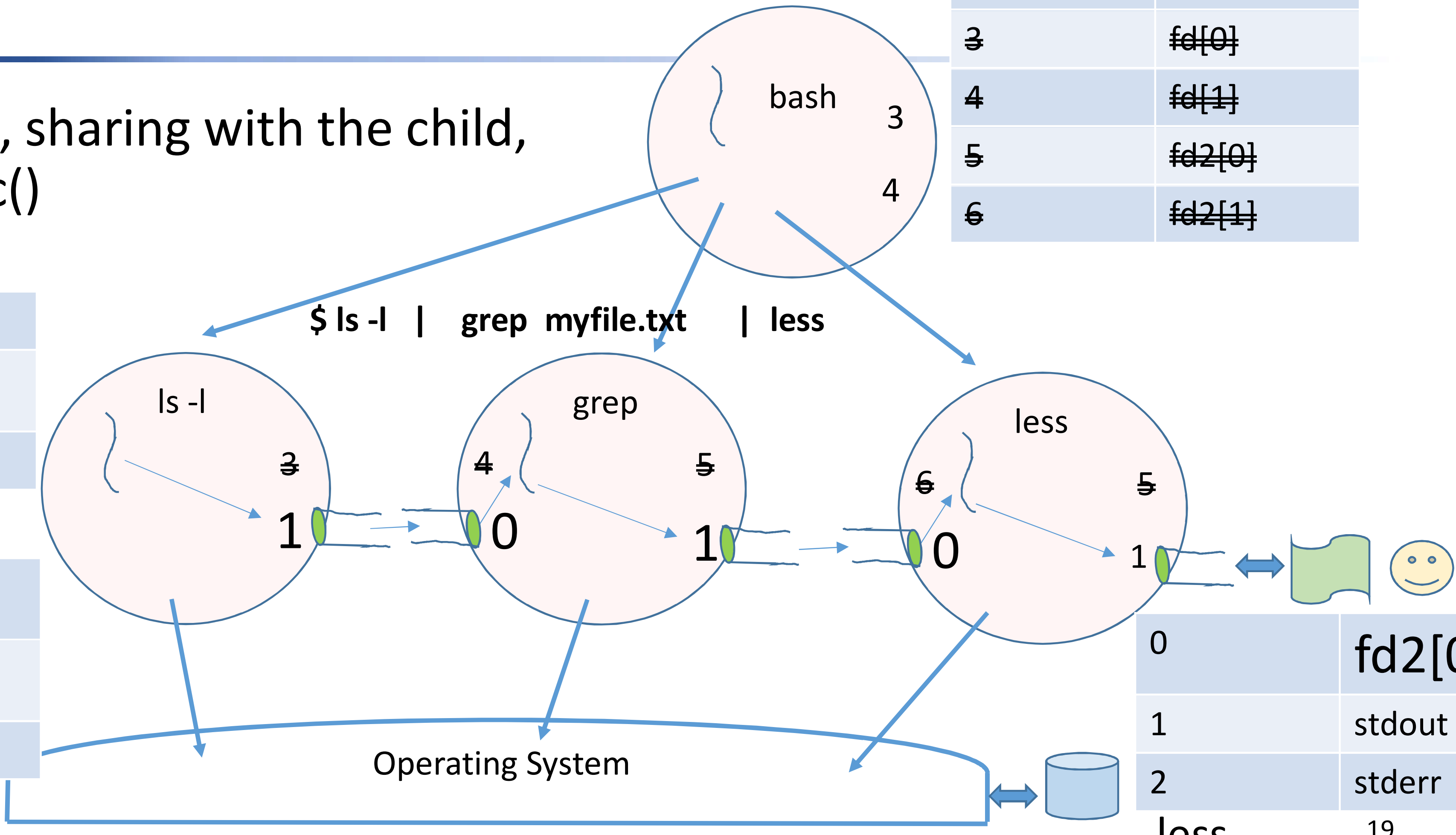
0	stdin
1	stdout
2	stderr
3	fd[0]
~~4~~	~~fd[1]~~

Example

- pipe (fd), sharing with the child, and exec()

bash

0	stdin
1	stdout
2	stderr
~~3~~	~~fd[0]~~
~~4~~	~~fd[1]~~
~~5~~	~~fd2[0]~~
~~6~~	~~fd2[1]~~



ls

0	stdin
1	fd[1]
2	stderr

grep

0	fd[0]
1	fd2[1]
2	stderr

less

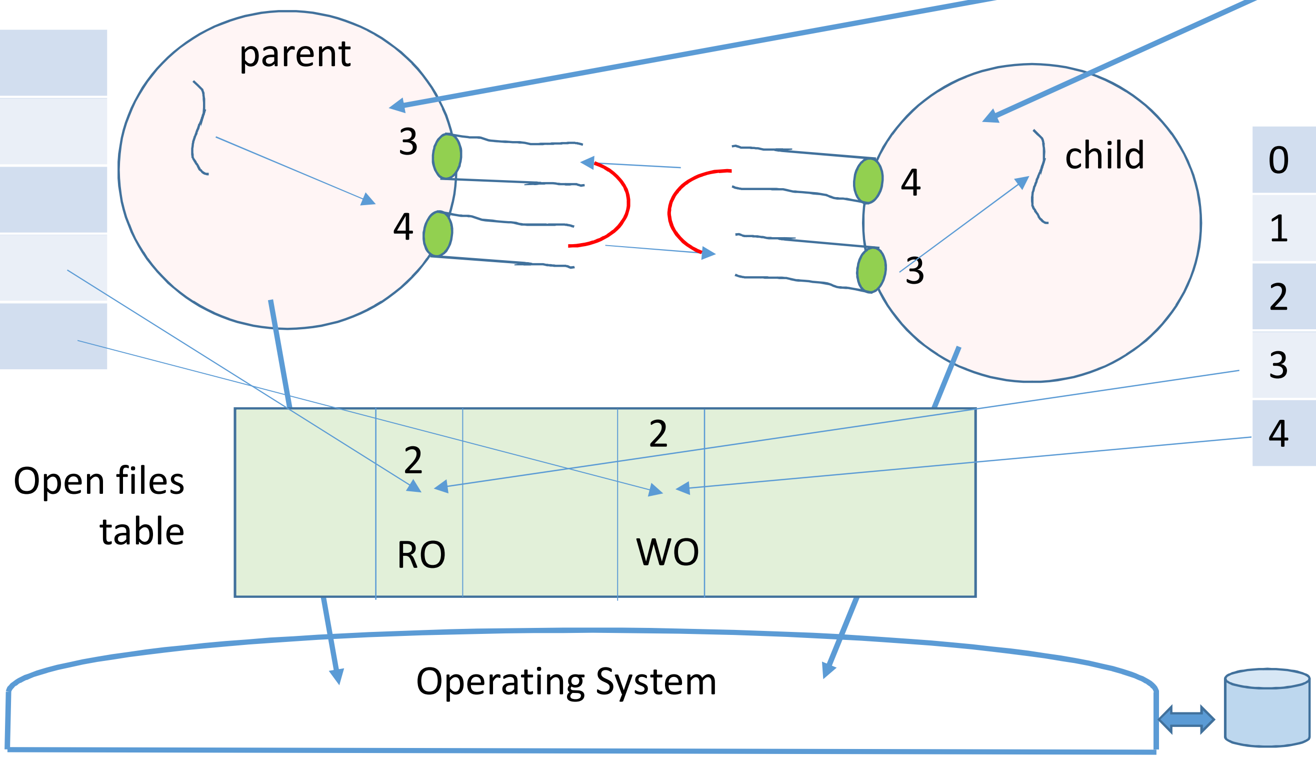
0	fd2[0]
1	stdout
2	stderr

The open files table

```
int fd[2];  // = {?,?}
pipe (fd);  // = {3,4}
int pid = fork();
```

- How it is possible to close all additional channels without closing effectively the pipe?

0	stdin
1	stdout
2	stderr
3	fd[0]
4	fd[1]



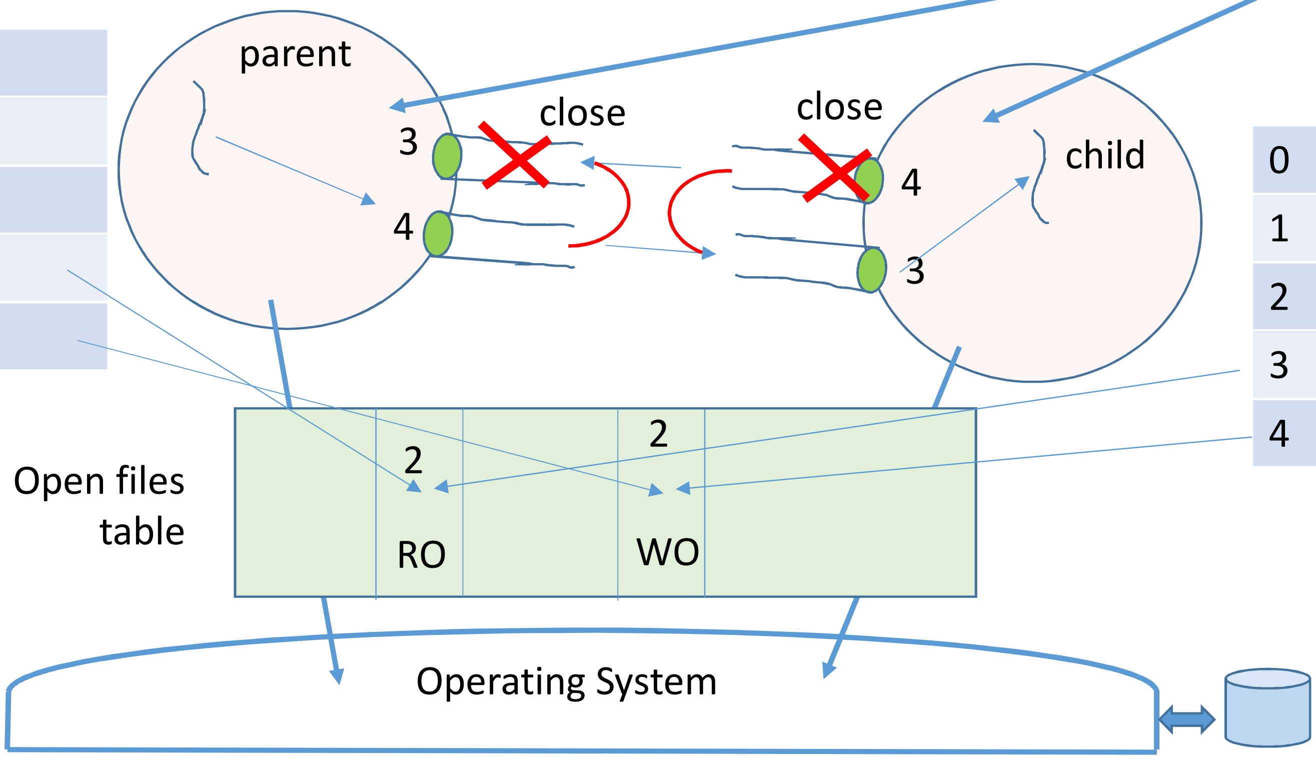
0	stdin
1	stdout
2	stderr
3	fd[0]
4	fd[1]

The open files table

```
int fd[2];   // = {?,?}
pipe (fd);   // = {3,4}
int pid = fork();
```

- How it is possible to close all additional channels without closing effectively the pipe?

0	stdin
1	stdout
2	stderr
3	fd[0]
4	fd[1]

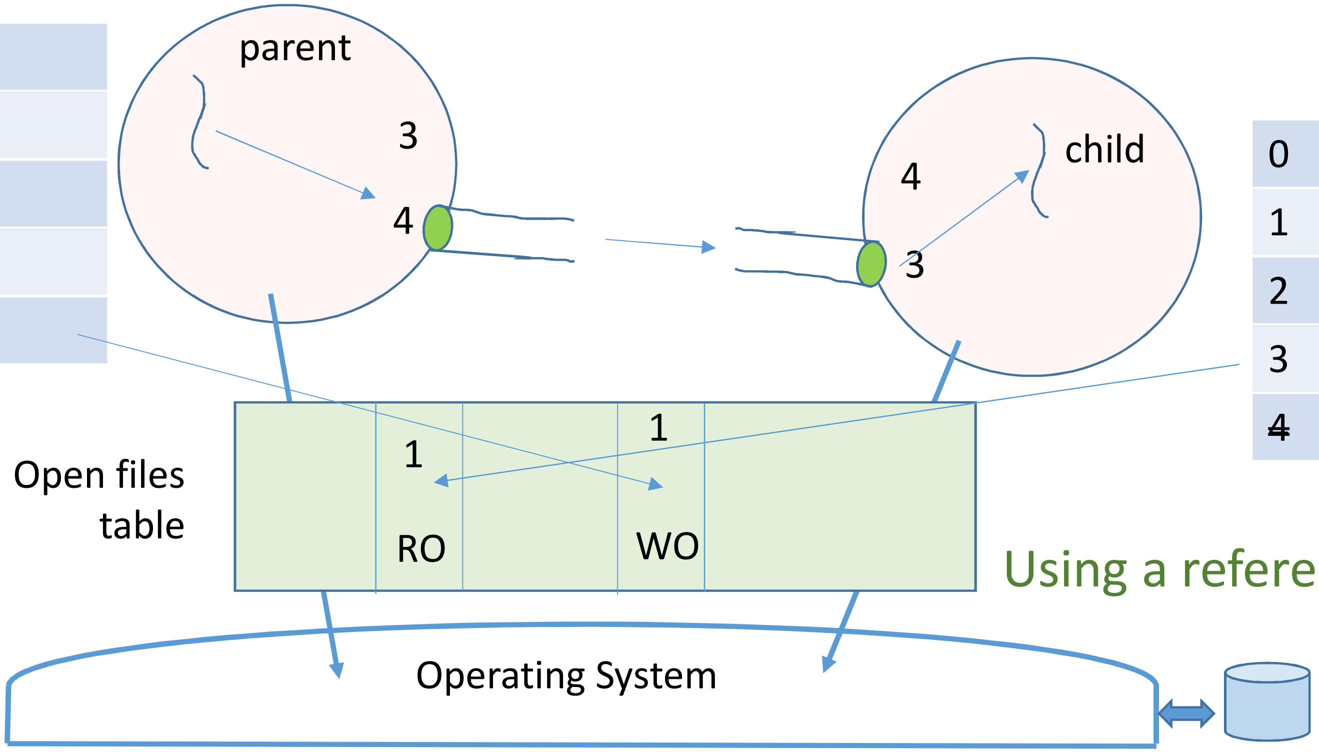


0	stdin
1	stdout
2	stderr
3	fd[0]
4	fd[1]

The open files table

- How it is possible to close all additional channels without closing effectively the pipe?

0	stdin
1	stdout
2	stderr
~~3~~	~~fd[0]~~
4	fd[1]



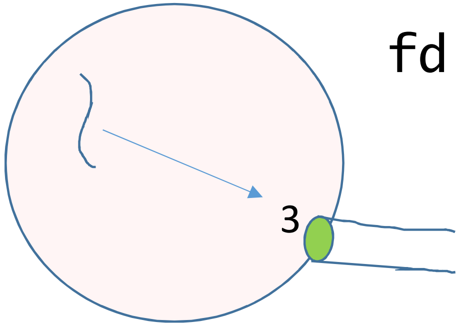
0	stdin
1	stdout
2	stderr
3	fd[0]
~~4~~	~~fd[1]~~

Using a reference counter

The open files table

- Each file descriptor points to the actual “open file”

```
fd = open (name, O_CREAT | O_TRUNC |
                              O_RDWR, 0640);
```

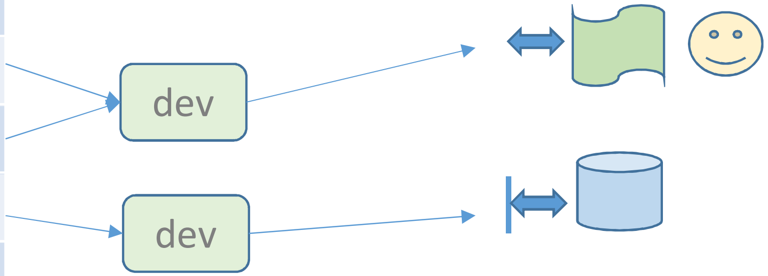


0	stdin
1	stdout
2	stderr
3	fd
4	-

Process file descriptor table

Ref count	flags	I/O ptr	devptr
0	-	-	-
1	R	876	console
2	W	1022	console
1	RW	0	disk
0	-		-

System open files table

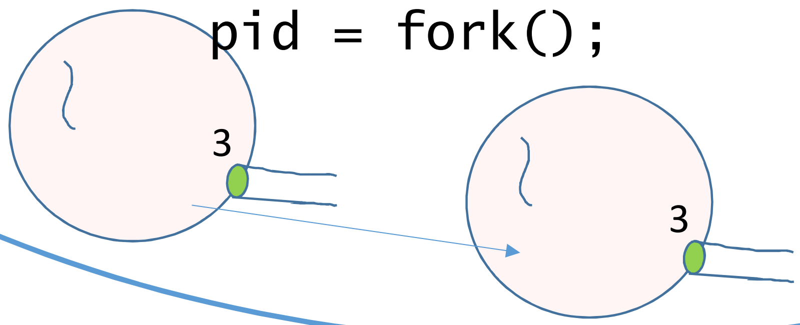


The open files table

- It is an OS table to manage all files being used in the system
- Fields
  - Pointer to the file *internal structure* (to the filesystem, usually on disk)
  - I/O pointer (lseek)
  - Open mode (read, write, read/write, append...)
  - Number of references (from channel tables)
    - Possibly from different processes
- I/O pointer is individual per open files entry
  - Shared among file descriptors that use the same entry – dup, fork...
  - Private for files opened with separate “open” calls

The open files table

- Each file descriptor points to the actual "open file"



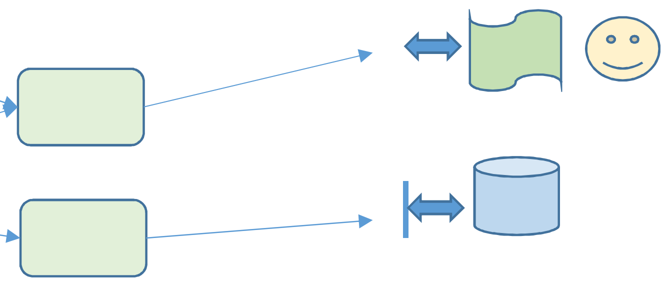
0	stdin
1	stdout
2	stderr
3	fd
4	-

0	stdin
1	stdout
2	stderr
3	fd
4	-

Process file descriptor table

Ref count	flags	I/O ptr	ptr
0	-	-	-
2	R	876	console
4	W	1022	console
2	RW	0	disk
0	-		-

System open files table



i-nodes

I/O System Calls

- Basic I/O System Calls
  - Fd = open(path, flags[, permissions]);
  - close(fd);
  - Bytes = read(fd, @ref, bytes);
  - Bytes = write(fd, @ref, bytes);
  - Newfd = dup(fd);
  - Newfd = dup2(fd, newfd);
  - *pipe(fd_vector);*

- Blocking vs Non-blocking System calls

Bibliography

- Operating system concepts (John Wiley & Sons, INC. 2014)
  - Silberschatz, A.; Galvin, P.B.; Gagne, G
  - http://cataleg.upc.edu/record=b1431631~S1*cat

- Operating systems: internals and design principles (Prentice Hall, 2015)
  - Stallings, W
  - http://cataleg.upc.edu/record=b1441252~S1*cat