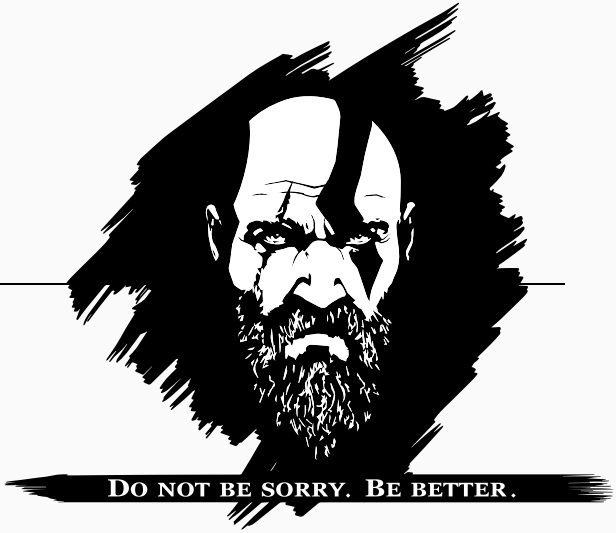
42sh - Presentation

ACU 2019 Team



Copyright

Rules

- You must have downloaded your copy from the Assistants' Intranet <https://intra.assistants.epita.fr>.
- This document is strictly personal and must **not** be passed on to someone else.
- Non-compliance with these rules can lead to severe sanctions.

Recap

Pipe in Shell

Definition

A pipeline is a sequence of one or more commands separated by the control operator '|'. The standard output of all but the last command shall be connected to the standard input of the next command. – SCL

Example

```
42sh$ echo "I love 42sh" | wc -c
12
```

Redirection if Shell

Definition

Redirection is used to open and close files for the current shell execution environment or for any command. – SCL

Example

```
42sh$ echo "I love 42sh" > my_file
42sh$ cat my_file
I love 42sh
```

pipe()

What is a pipe?

Definition

pipe() creates a pipe, a unidirectional data channel that can be used for interprocess communication. – man 2 pipe

Usage

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

- You can **write** data on fd[1]
- You can **read** those data on fd[0]

What about fork()?

```c
int fd[2];
pipe(fd);
pid_t pid = fork();
if (pid == 0)
{
    // Child fd
}
else
{
    // Parent fd
}
```

- Both process can write or read in the pipeline
- We don't want to do that

Parents write to child

```c
int fd[2];
pipe(fd);
pid_t pid = fork();
if (pid == 0)
{
    close(fd[1]); // Close unused write entrance
    // Read from fd[0]
    close(fd[0]); // Close when finished
}
else
{
    close(fd[0]); // Close unused read entrance
    // Write to fd[1]
    close(fd[1]); // Close when finished
}
```

- All writer **must** be closed
- What if child want to answer?

Two pipes!

```
pipe(parent_to_child);
pipe(child_to_parent);
if (fork() == 0)
{
    close(parent_to_child[1]); // Close unused write entrance
    close(child_to_parent[0]); // Close unused read entrance
    // Read from parent_to_child[0], write to child_to_parent[1]
    close(parent_to_child[0]); // Close when finished
    close(child_to_parent[1]); // Close when finished
}
else
{
    close(parent_to_child[0]); // Close unused read entrance
    close(child_to_parent[1]); // Close unused write entrance
    // Read from child_to_parent[0], write to parent_to_child[1]
    close(parent_to_child[1]); // Close when finished
    close(child_to_parent[0]); // Close when finished
}
```

dup2()

What's dup2?

Definition

The dup2() system call creates a copy of the file descriptor oldfd, using the file descriptor number specified in newfd. If the file descriptor newfd was previously open, it is silently closed before being reused. – man 2 dup

Example

```c
printf("First print\n");
int fd = open("output.txt",  O_RDWR | O_CREAT);
dup2(fd, STDOUT_FILENO);
printf("Second print\n");
```

- What happened if we *cat* `example.txt` ?

Redirection

The list

- [n]>
- [n]<
- [n]>>
- [n]<<
- [n]<<-
- [n]>&
- [n]<&
- [n]>|
- [n]<>

Just a question of flag

> and <

Just dup2() the fd of the opened file parameter with STDOUT or STDIN

>> and <<

Same but you must open the file with O_APPEND

<< and <<-

Heredoc! Put the doc in a tmp file, and open it later

The others

Read the SCL.

Piping

What we want to handle

```
ls | cat -e | wc
```

- Each | is a pipe()
- You have to redefine standard input and output

How to Ado that with just two processes?

With multiple pipe

```
def recursive_pipe(node, input_fd):
    fd = [ -1, STDOUT ] # If it's the last
    if (node.has_next()):
        fd = create_new_pipe()
    if (fork() == 0):
        set_input(input_fd)
        set_output(fd[write])
        exec(node)
    else:
        wait()
        recursive_pipe(node.get_next(), fd[read])
```