OS 2024 Lab 2

shell

Due Date: 2024/11/01 17:00 (before lab2 course finishes)

TA : P76121526 李政融 , VX6122035 曾遠哲

1. Introduction

2. Requirements

3. Grading

1.1 Shell introduction

1.2 Basic functionality

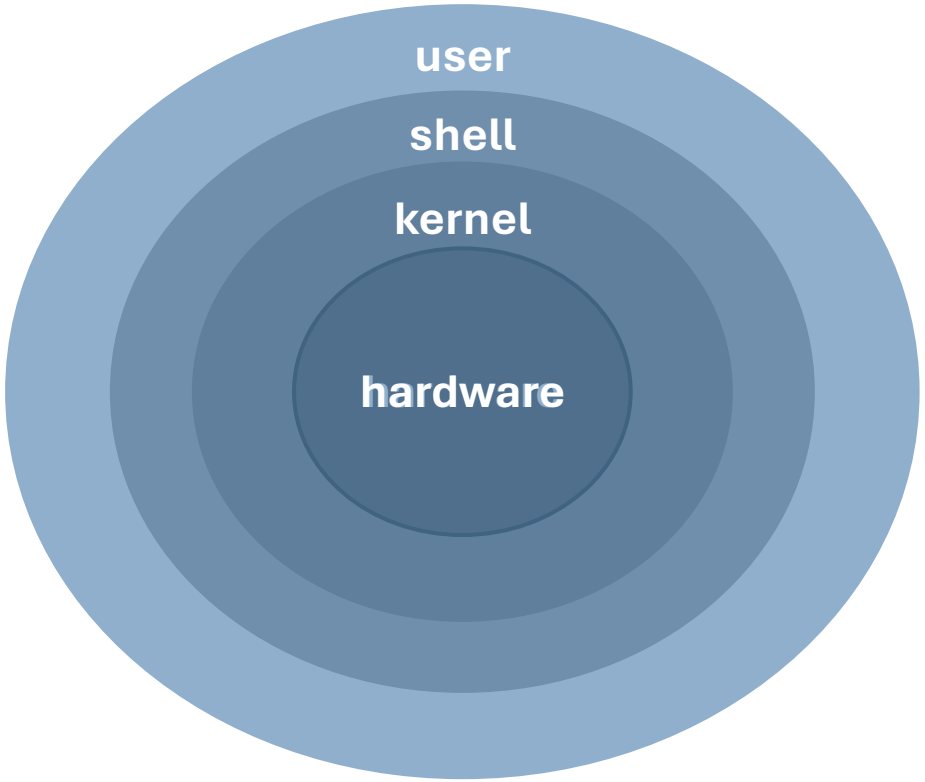
- Built-in commands
- External commands

1.3 Advanced functionality

- Redirection
- Pipe

Shell

It is an intermediary between the user and the kernel, allowing the user to control the computer through commands.



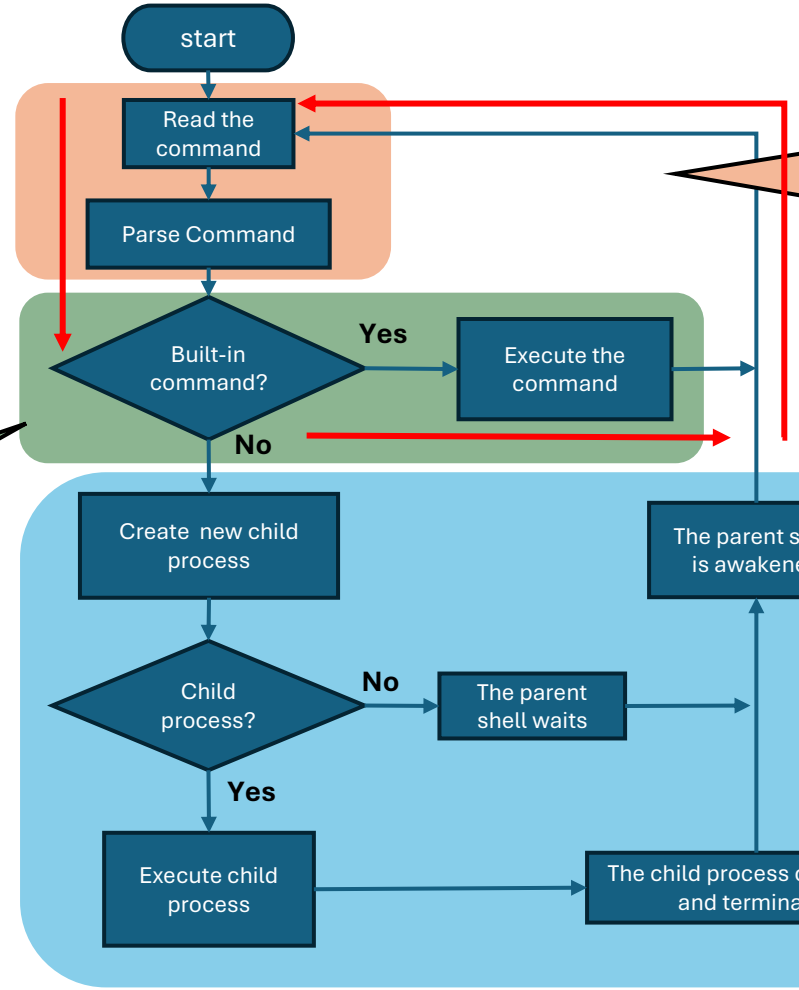
System architecture diagram

Flow diagram - basic functionality

Command : cd Desktop/oslab/

```
timatm@timatm-VirtualBox:~$ cd Desktop/oslab/
timatm@timatm-VirtualBox:~/Desktop/oslab$
```

Built-in command



Parse the command given by the user

Determine whether it is a built-in command. If so, execute the built-in function.

If it is an external command, call fork() and exec() to execute the external command.

Flow diagram - basic functionality

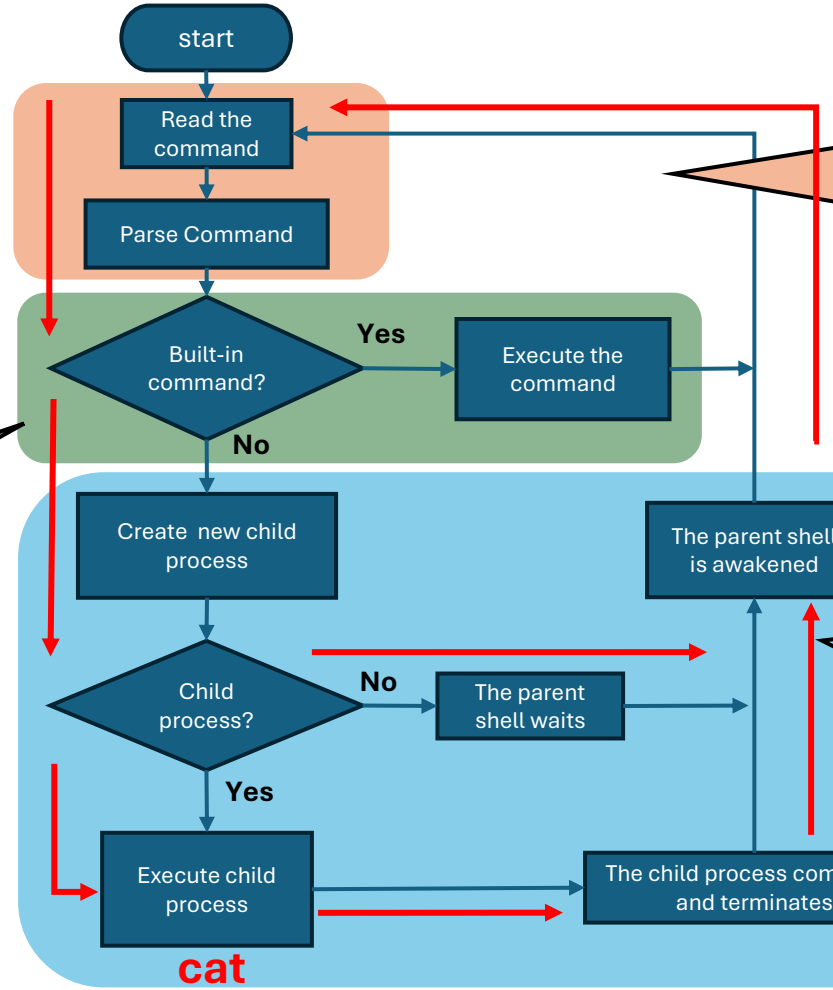
Command : cd Desktop/oslab/

```
timatm@timatm-VirtualBox:~$ cd Desktop/oslab/
timatm@timatm-VirtualBox:~/Desktop/oslab$
```

Command : cat test.txt

```
timatm@timatm-VirtualBox:~/Desktop/oslab$ cat test.txt
I love OS lab
```

External command



Parse the command given by the user

Determine whether it is a built-in command. If so, execute the built-in function.

If it is an external command, call fork() and exec() to execute the external command.

What are Built-in commands?

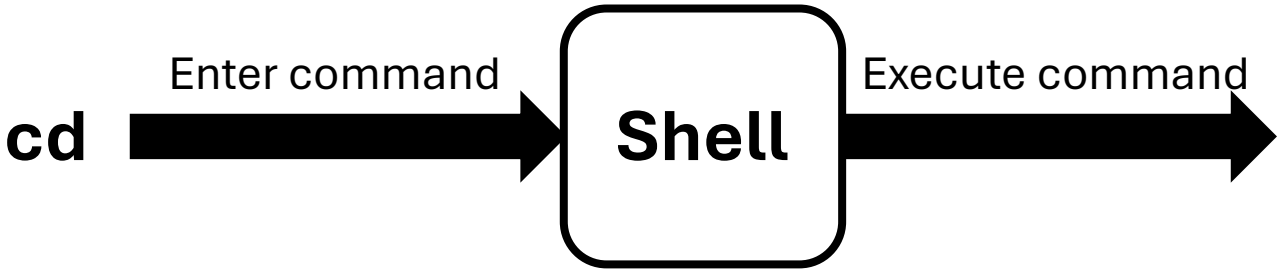
- Part of the shell (Bash, Zsh, etc.)
- Executed by the shell, no new process created
- Generally efficient and faster than external commands
- Commands like cd, echo, pwd, and exit.

What are External commands?

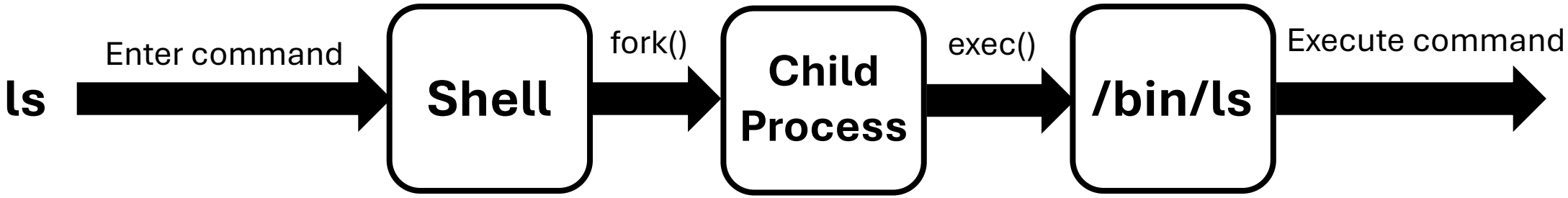
- Programs outside the shell (like /bin/ls, /usr/bin/find, etc.)
- Executed by the shell, but new processes are created to run them
- Typically slower than built-in commands due to process creation overhead
- Commands like ls, grep and cat

Command type

- Built-in Command



- External Command



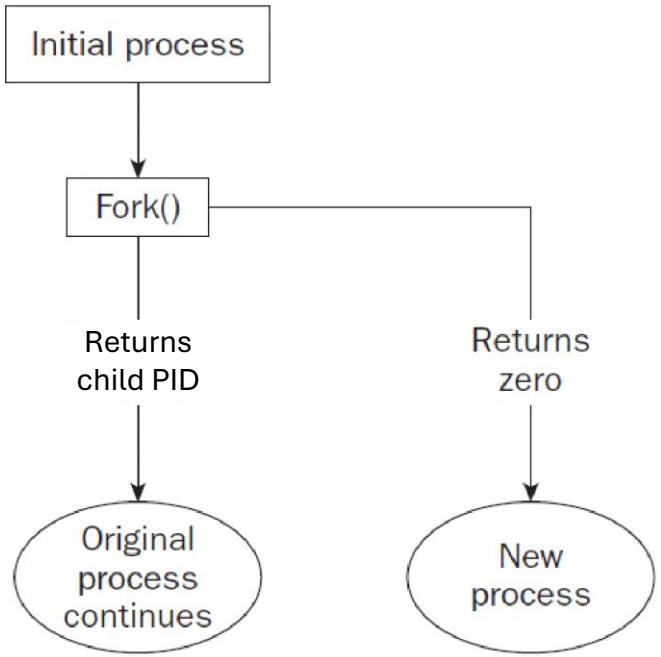
Fork

```
int
main(void)
{
    pid_t pid;

    if (signal(SIGCHLD, SIG_IGN) == SIG_ERR) {
        perror("signal");
        exit(EXIT_FAILURE);
    }
    pid = fork();
    switch (pid) {
    case -1:
        perror("fork");
        exit(EXIT_FAILURE);
    case 0:
        puts("Child exiting.");
        exit(EXIT_SUCCESS);
    default:
        printf("Child is PID %jd\n", (intmax_t) pid);
        puts("Parent exiting.");
        exit(EXIT_SUCCESS);
    }
}
```

child process

parent process



Fork() diagram

Execvp

Using execvp()

The following example searches for the location of the *ls* command among the directories specified by the *PATH* environment variable, and passes arguments to the *ls* command in the *cmd* array.

```
#include <unistd.h>

int ret;
char *cmd[] = { "ls", "-l", (char *)0 };
...
ret = execvp ("ls", cmd);
```

NOTE :

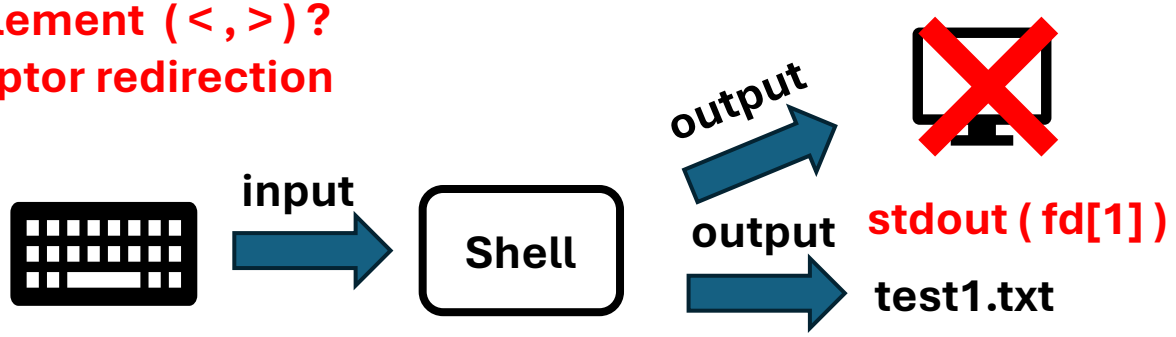
- **In order to allow students to practice fork(), the use of system() is prohibited in this lab.**

Shell advanced functionality

How to implement (< , >) ?
→ file descriptor redirection

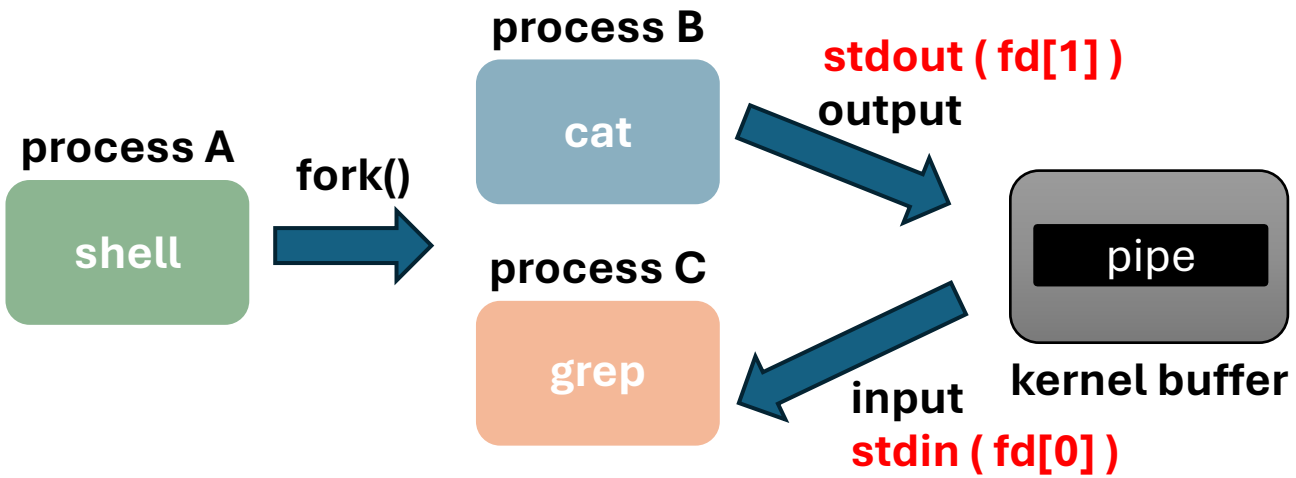
Command : cat test.txt > test1.txt

```
timatm@timatm-VirtualBox:~/Desktop/oslab$ cat test.txt > test1.txt
timatm@timatm-VirtualBox:~/Desktop/oslab$ cat test1.txt
I love OS lab
```



Command : cat test1.txt | grep lab

```
timatm@timatm-VirtualBox:~/Desktop/oslab$ cat test1.txt | grep lab
I love OS lab
```

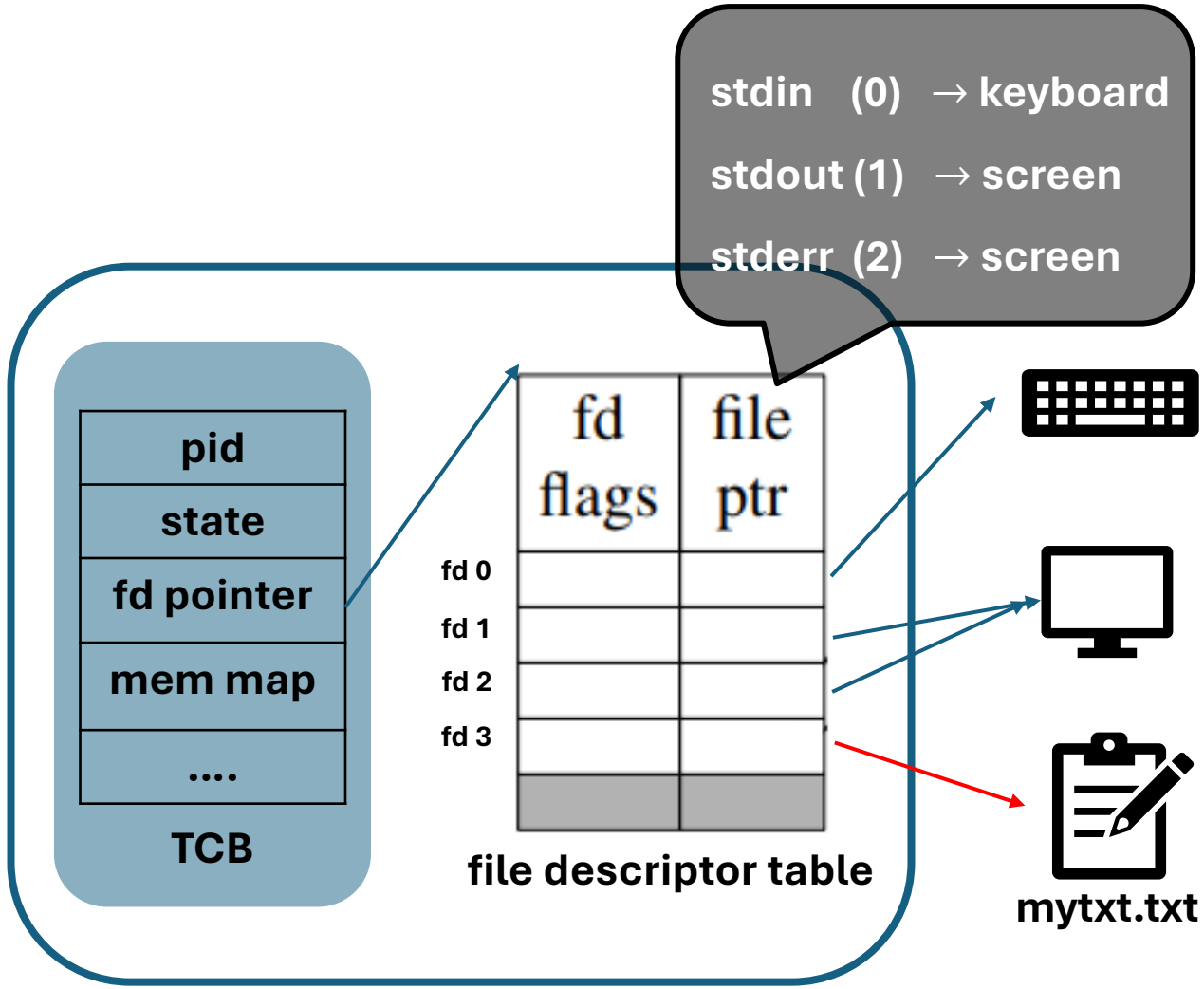


File descriptor

Task Control Block (TCB):
A data structure used by the OS to manage processes/threads.

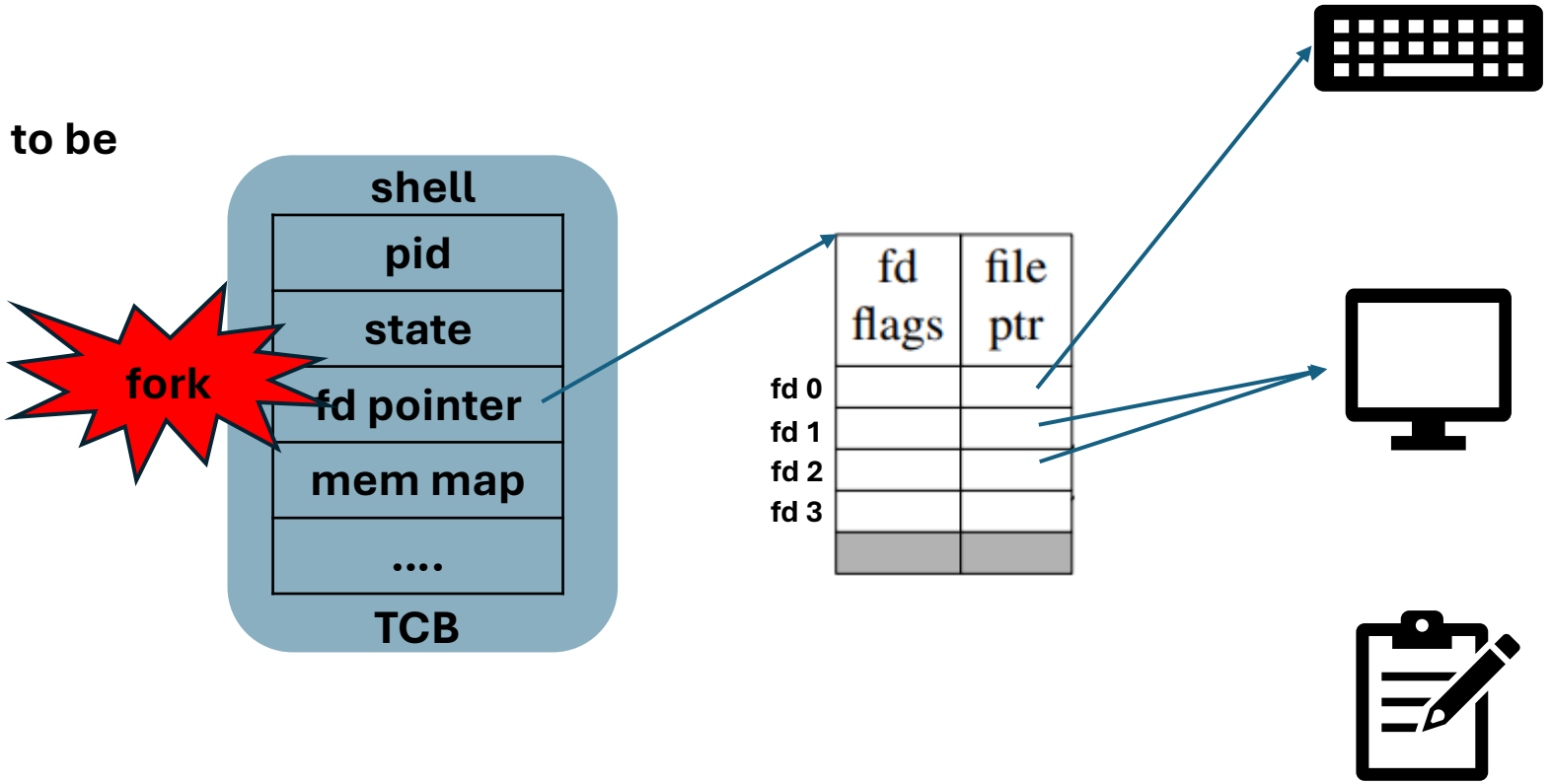
File descriptor table:
A table unique to each process, mapping file descriptors (small integers) to open files, sockets, or other resources.

Open a file (mytxt.txt)



File descriptor (after fork())

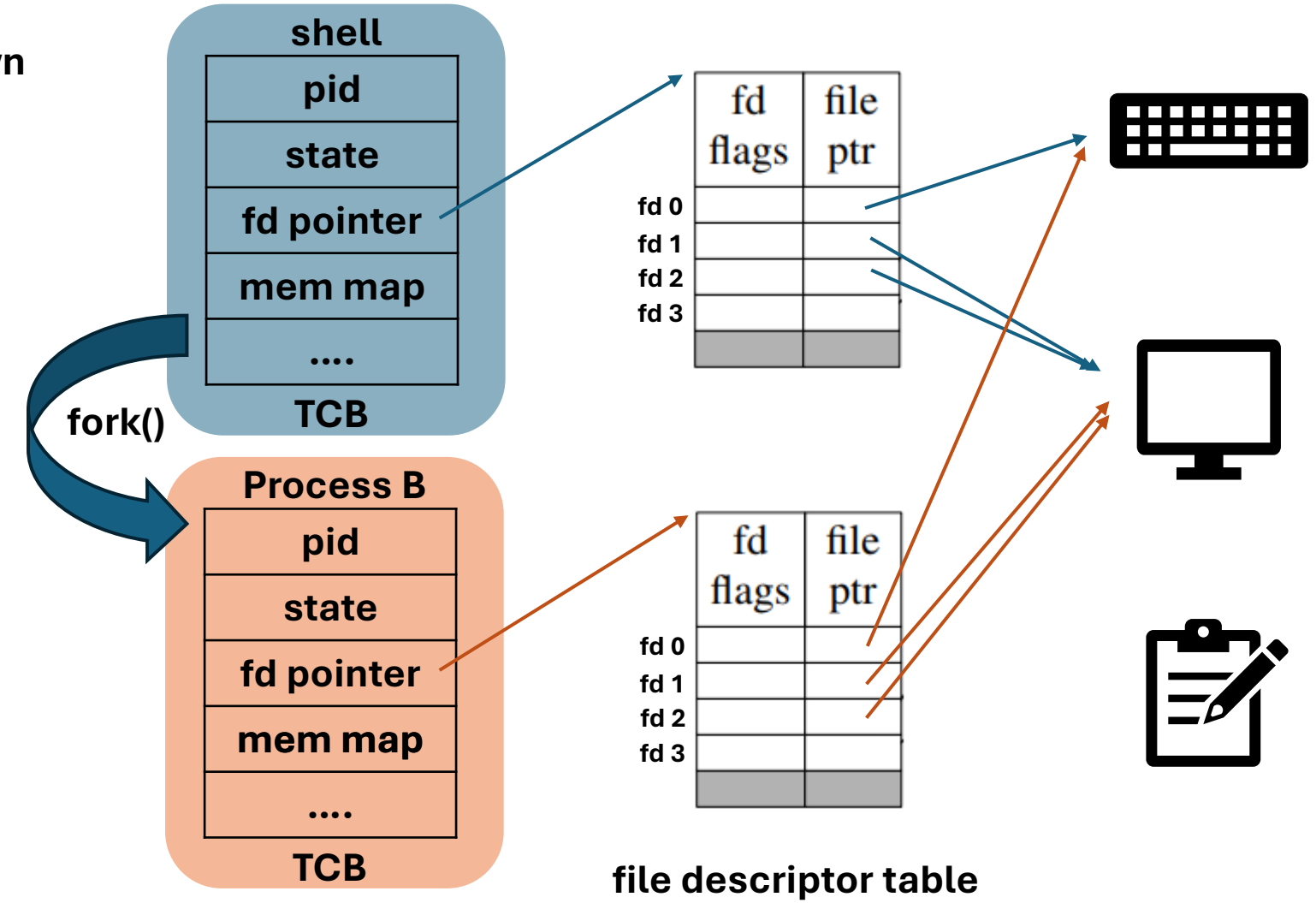
- The right side is the architecture of the original shell
- When an external command is to be executed, fork() is called



file descriptor table

File descriptor (after fork())

- The child process creates its own TCB and file descriptor table

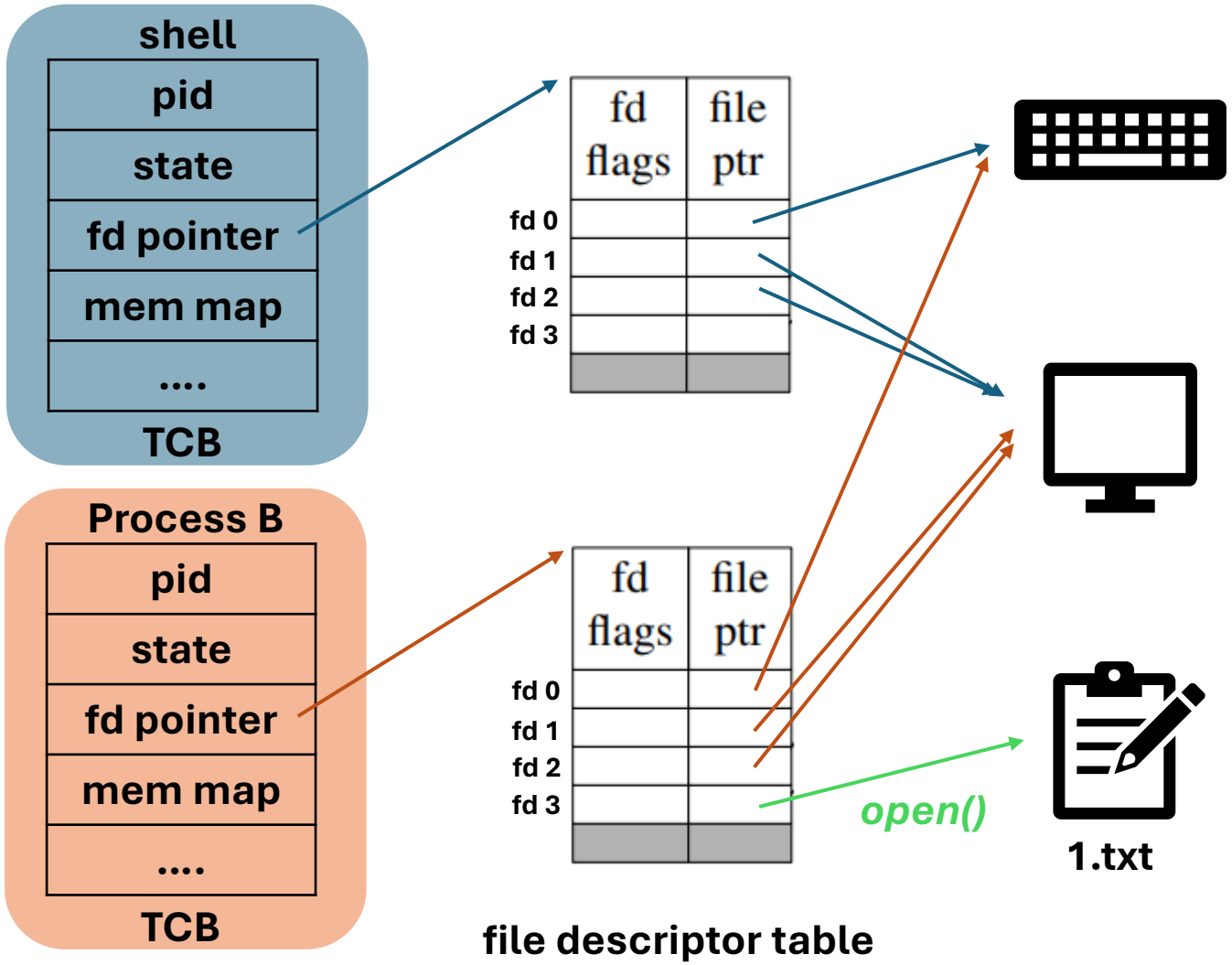


Redirection

Implementation redirection (>) operation :

ls > 1.txt

- **Open the file that needs to be redirected**
- **Use dup2() to redirect the stdin and stdout of the process**



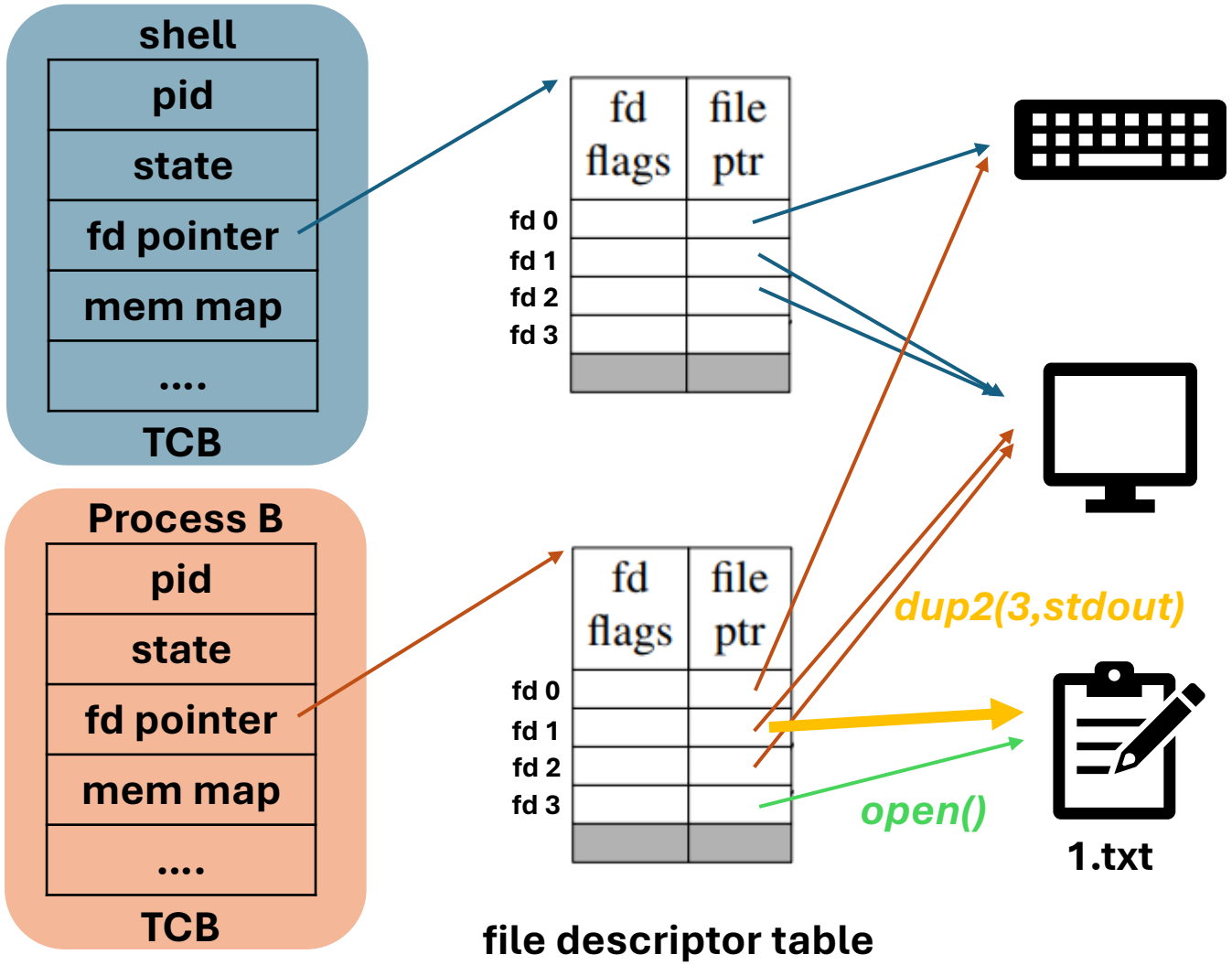
file descriptor table

Redirection

Implementation redirection (>) operation :

ls > 1.txt

- **Open the file that needs to be redirected**
- **Use dup2() to redirect the stdin and stdout of the process**



file descriptor table

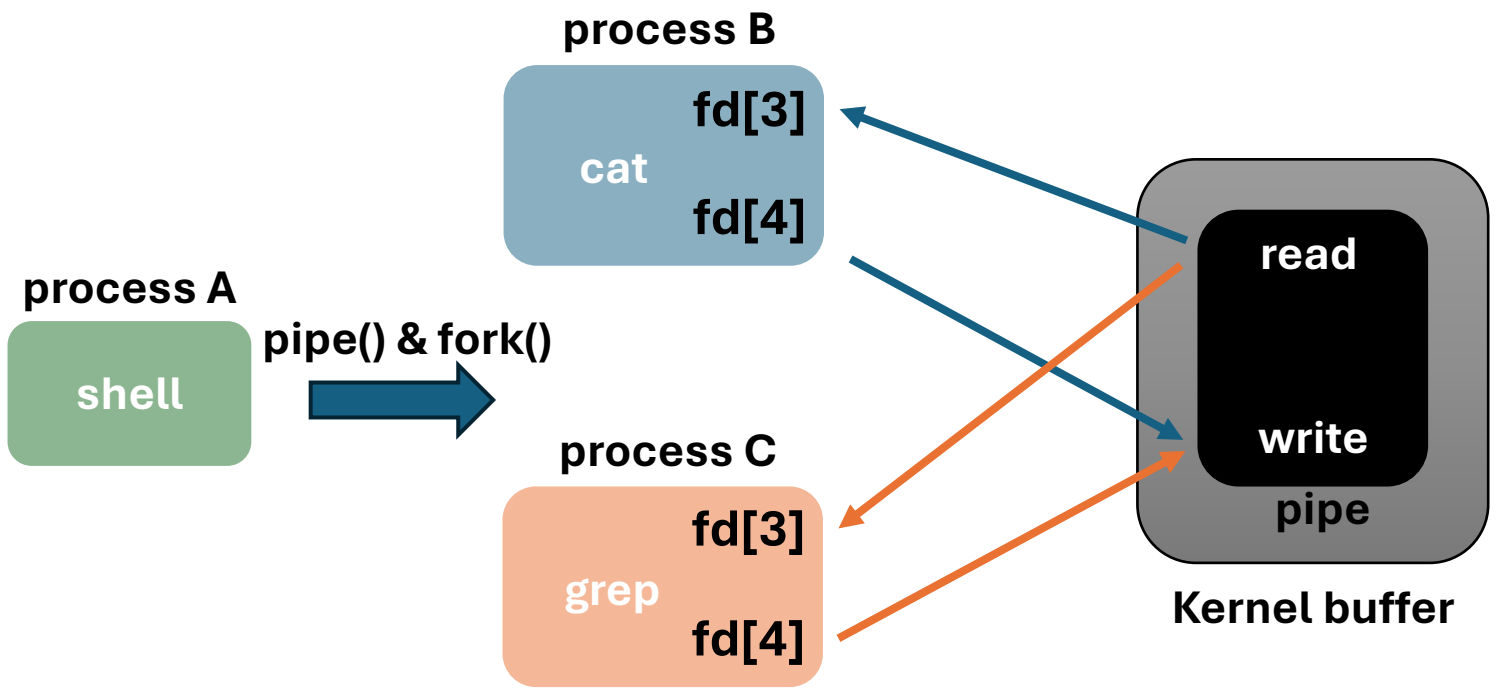
Pipe

Implementation pipe (|) operation :

cat test1.txt | grep lab

- Use pipe() and fork() to point the file descriptor to read and write respectively
- Close unused file descriptors
- Use dup2() to redirect stdin and stdout

After pipe() and fork()?



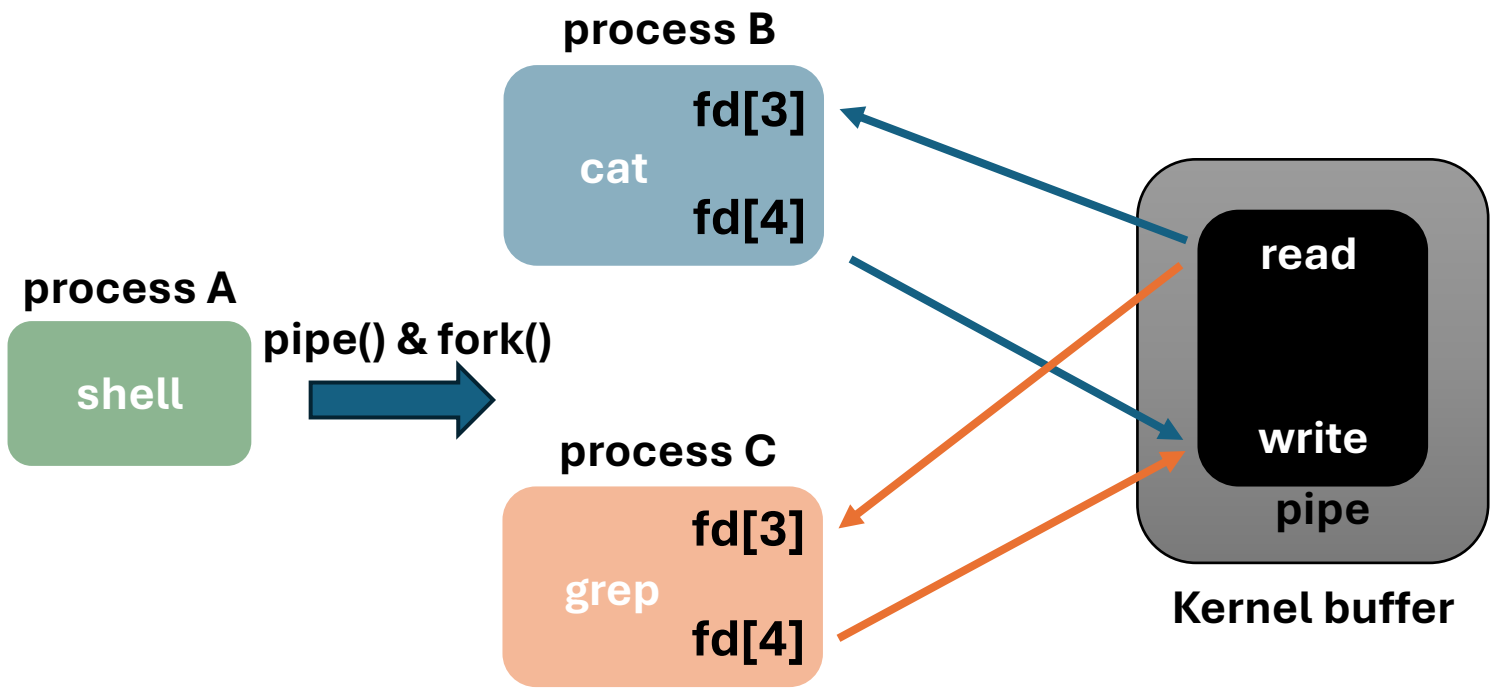
Pipe

Implementation pipe (|) operation :

cat test1.txt | grep lab

- Use pipe() and fork() to point the file descriptor to read and write
- Close unused file descriptors
- Use dup2() to redirect stdin and stdout

Close redundant mapping
→ Close read of processB
→ Close write of processC

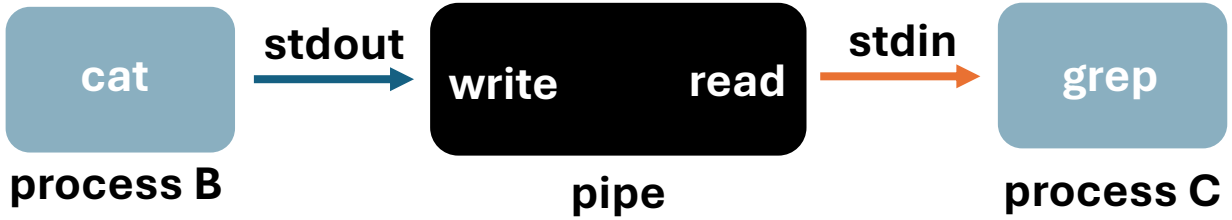


Pipe

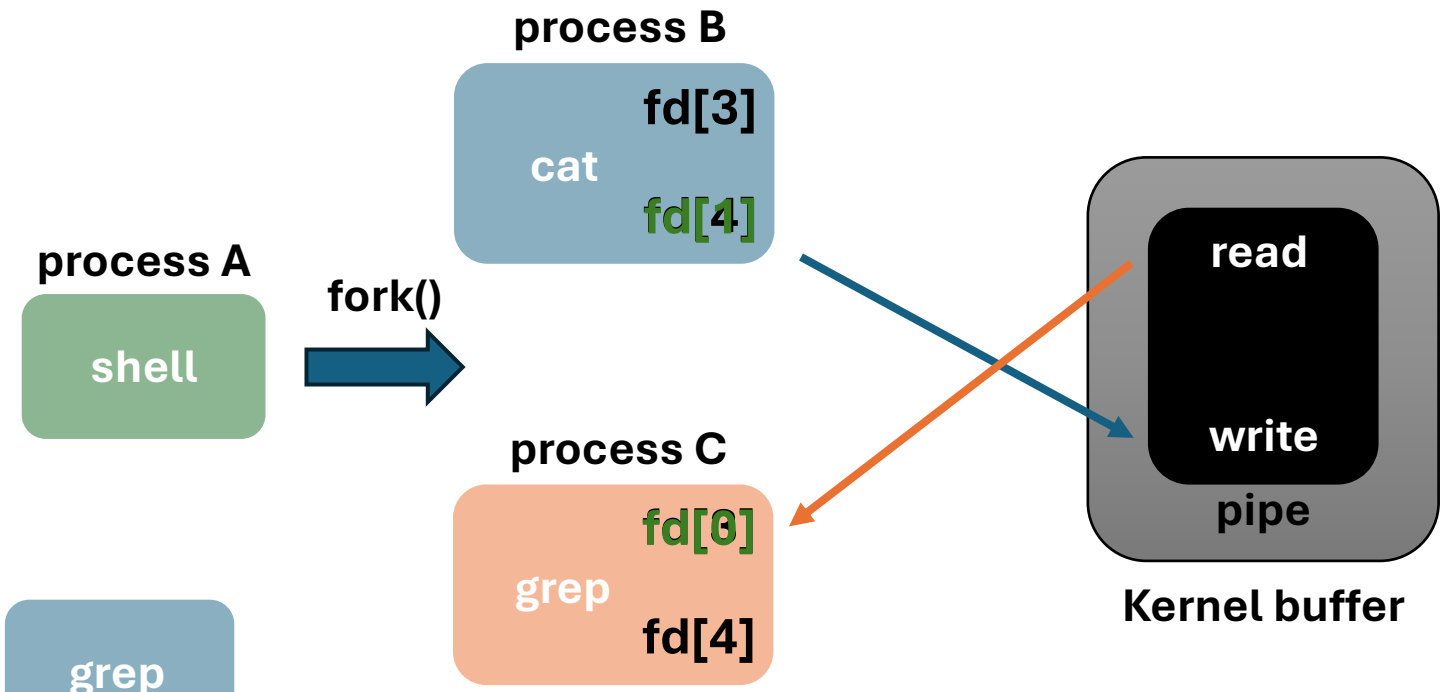
Implementation pipe (|) operation :

cat test1.txt | grep lab

- Use pipe() and fork() to point the file descriptor to read and write
- Close unused file descriptors
- Use dup2() to redirect stdin and stdout



After redirection?



1. Introduction

2. Requirements

3. Grading

2.1 Built-in command cd implementation

2.2 External commands implementation

2.3 Redirection implementation

2.4 Pipe implementation (bonus)

Input format

1. **Only 3 special operators: | , < and > .**
   - No quotation marks(" or '), e.g., "string", 'string'
2. **All the cmds, args, operators will be separated by space char.**
   - 指令(cmd), 引數(arg),特殊符號(operator) 都會用空白符號隔開
3. **Input redirection (<) only show up after last command.**
   - Input redirection 的檔名一定會接在 < 後面，且如果有，一定會緊接在第一個指令後面
4. **Output redirection (>) only show up after last command.**
   - Output redirection 的檔名一定會接在 > 後面，且如果有，一定會緊接在最後一個指令後面

格式	>>> $ cmd args < infile | cmd args | cmd args > outfile
範例 1	>>> $ cd Desktop/oslab/
範例 2	>>> $ cat test1.txt > out.txt
範例 3	>>> $ cat test1.txt | grep lab

Data structure

```
struct cmd_node {
  char **args;
  int length;
  char *in_file, *out_file;
  int in ,out
  struct cmd_node *next;;
}
```

```
struct cmd {
    struct cmd_node *head;
    int pipe_num;
}
```

```
>>> $ cd Desktop/oslab/
```

cmd

head → cmd_node

pipe_num = 1

cmd_node

args[0] = cd

args[1] = Desktop/oslab/

length = 2

in_file = out_file = NULL

in = 0 , out = 1

next = NULL

```
>>> $ cat test1.txt > out.txt
```

cmd

head → cmd_node

pipe_num = 1

cmd_node

args[0] = cat

args[1] = test1.txt

length = 2

in_file = NULL

out_file = out.txt

in = 0 , out = 1

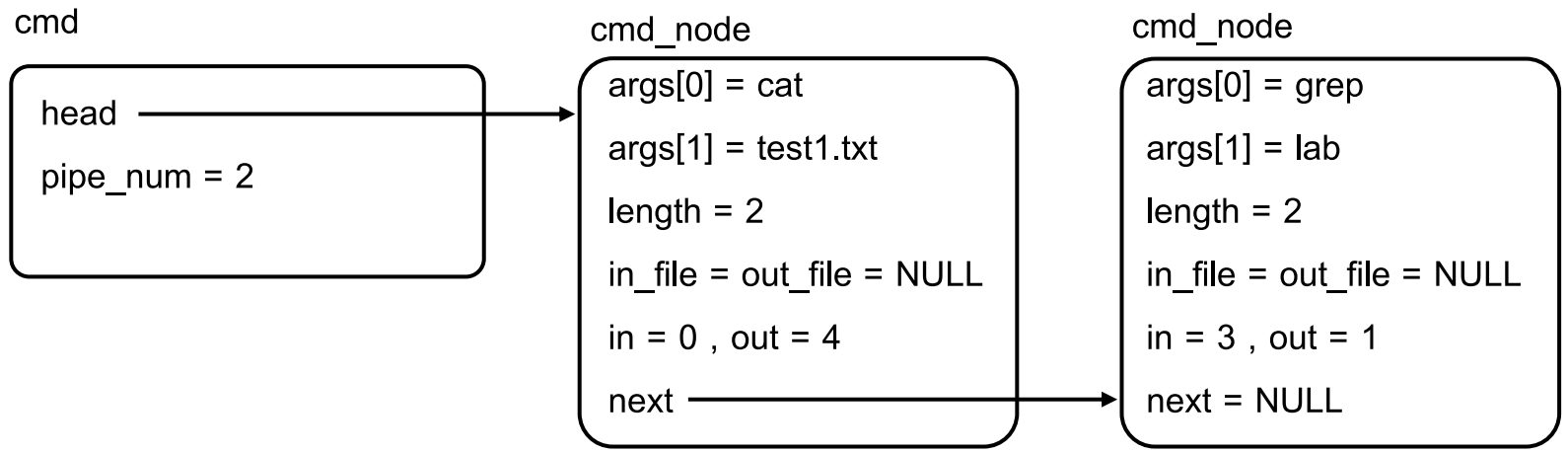
next = NULL

Data structure

```
struct cmd_node {
  char **args;
  int length;
  char *in_file, *out_file;
  int in ,out
  struct cmd_node *next;
}
```

```
struct cmd {
    struct cmd_node *head;
    int pipe_num;
}
```

```
>>> $ cat test1.txt | grep lab
```



2.1 Built-in command cd implementation

- **Objective:** Complete the **cd** function so the program can execute this built-in command, and you can take **pwd** function as the reference.
  - **NAME** cd - change the working directory
    SYNOPSIS cd [directory]
  - **NAME** pwd - print name of current working directory
    SYNOPSIS pwd
- **Function to complete:**
  - cd() in /src/builtin.c

```
oslab@PC:~$ cd ./path/to/directory
oslab@PC: path/to/directory$
```

```
oslab@PC: path/to/directory$ pwd
/home/path/to/directory
```

2.1 Built-in command cd implementation

- **Test case & expected result**

```
>>> $ pwd
/oslab2
>>> $ cd ./shell
>>> $ pwd
/oslab2/shell
>>> $
```

2.2 External command implementation

- **Objective:** The **spawn_proc** function forks a child process to execute an external command, while the parent process waits for the child to finish.

- **Function to complete:**
  - spawn_proc() in shell.c

- **Test case & expected result**

```
>>> $ ls
builtin.o  demo.txt  makefile  my_shell.c  src
command.o  include   my_shell  shell.o
>>> $
```

```
>>> $ cat test1.txt
Today is os' Day.
I am a student in CSIE.
I love os, you love os.
I am going to score 100 point.
Have a nice os' Day.
>>> $
```

2.3 Redirection implementation

- **Objective:** Complete **the redirection** function to manage input and output redirection for both built-in and external commands.

- **Function to complete:**
  - redirection() in shell.c

- **Test case & expected result**

```
>>> $ cat test1.txt > out.txt
>>> $ cat out.txt
Today is os' Day.
I am a student in CSIE.
I love os, you love os.
I am going to score 100 point.
Have a nice os' Day.
>>> $
```

2.4 Pipe implementation (bonus)

- **Objective:** Complete the **fork_cmd_node** function to manage the execution of multiple commands connected by pipes. Ensure proper handling of data flow and error management between the processes.

- **Function to complete:**
  - fork_cmd_node() in shell.c

- **Test case & expected result**

```
>>> $ cat test1.txt | tail -2
I am going to score 100 point.
Have a nice os' Day.
>>> $
```

1. Introduction

2. Requirements

3. Grading

requirement	points
2.1 Built-in command cd implementation	**2**
2.2 External command	**4**
2.3 Redirection for external commands	**4**
2.4 Pipe implementation (bonus)	**1**

Precautions

- Due Date: **2024/11/01 17:00 (before lab2 course finishes)**
- You should implement lab2 with **C** language.
- You will get **two folders (include & src)** and **three files (demo.txt & makefile & my_shell.c)** from **os_2024_lab2_template**
- You can modify makefile as you want, but make sure your **makefile** can compile your codes and create the executable successfully.