Lecture 7: Pointer wrap up & Processes

CSE 29: Systems Programming and Software Tools

Olivia Weng

Announcements

- Sign up for Exam 1 on prairietest.com

- Problem set 2 will be released today

Pointers allow us to pass around arrays

```
// Take a 2-byte char array of a UTF-8 encoding
// and produce an integer for that code point
uint16_t decode2(char encoding[]);


// Test case: é contains codepoint 233
```

é = 2bytes UTF-8

↳ 233

$$\underbrace{110\overbrace{xxxxx}}_{\text{1st byte}} \quad \underbrace{10\overbrace{xxxxxx}}_{\text{2nd byte}}$$

$$\downarrow$$

$$\underbrace{xxxxxxxxxxx}_{\text{code point}}$$

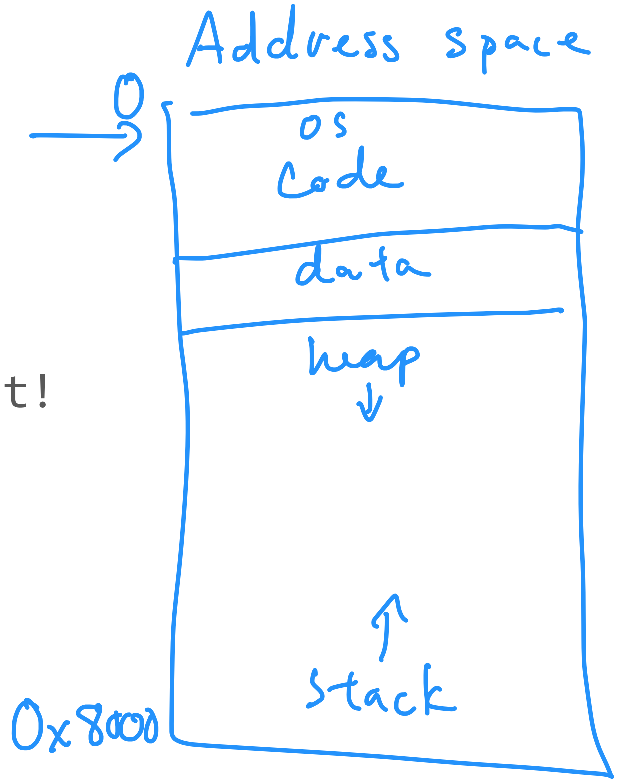
where 233 is contained

What if my pointer points to nothing?

- A pointer points to address 0

```
int *p = 0x0;

printf("%d \n", *p); // segmentation fault!
```



What is a segmentation fault?

- segfault: when a program tries to access memory it is not allowed to access.

- When does this happen?
  - Dereferencing a null pointer
  - Dereferencing a pointer the program does not have access to

Should I panic when I segfault?

- No! Instead:
    - Look at the pointers you are working with
    - print them with %p
    - If it's 0:
        - likely dereferencing null pointer
    - If it's close to 0:
        - likely dereferencing a pointer you don't have access to

- How to fix?
    - Back trace and see how the pointer's value and/or address was changed

What is char *argv[] in main()?

```
int main(int argc, char *argv[]);
```

$\underbrace{\texttt{argc}}_{\text{\# command line args}}$ $\underbrace{\texttt{char *argv[]}}$ $\longrightarrow$ array of char *

$\downarrow$

command line args

```
> ./demo7 hi cse29
```

$\underbrace{\texttt{./demo7}}_{\texttt{argv[0]}}$ $\underbrace{\texttt{hi}}_{1}$ $\underbrace{\texttt{cse29}}_{2}$

argv[0] = "./demo7"
argv[1] = "hi"
argv[2] = "cse29"

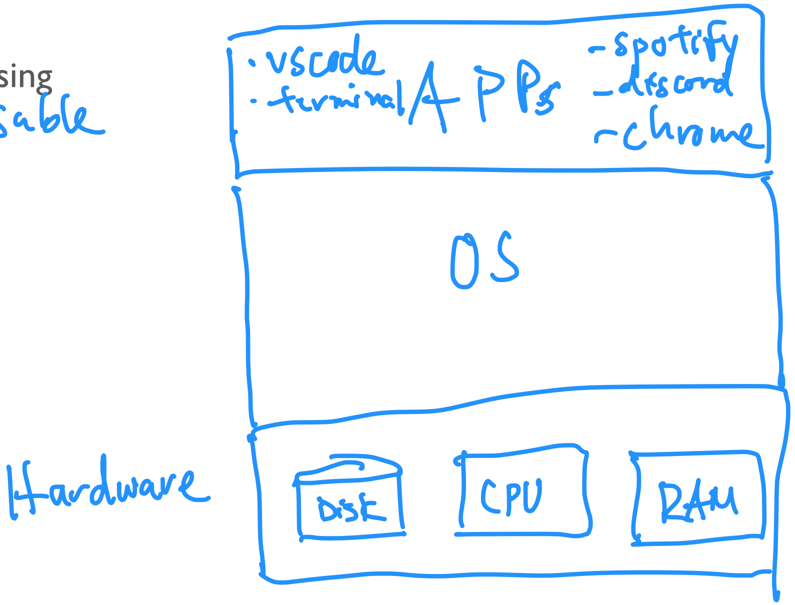
Demo

- `print_argv()`

What is an OS?

Examples: Linux
iOS
Android
Windows

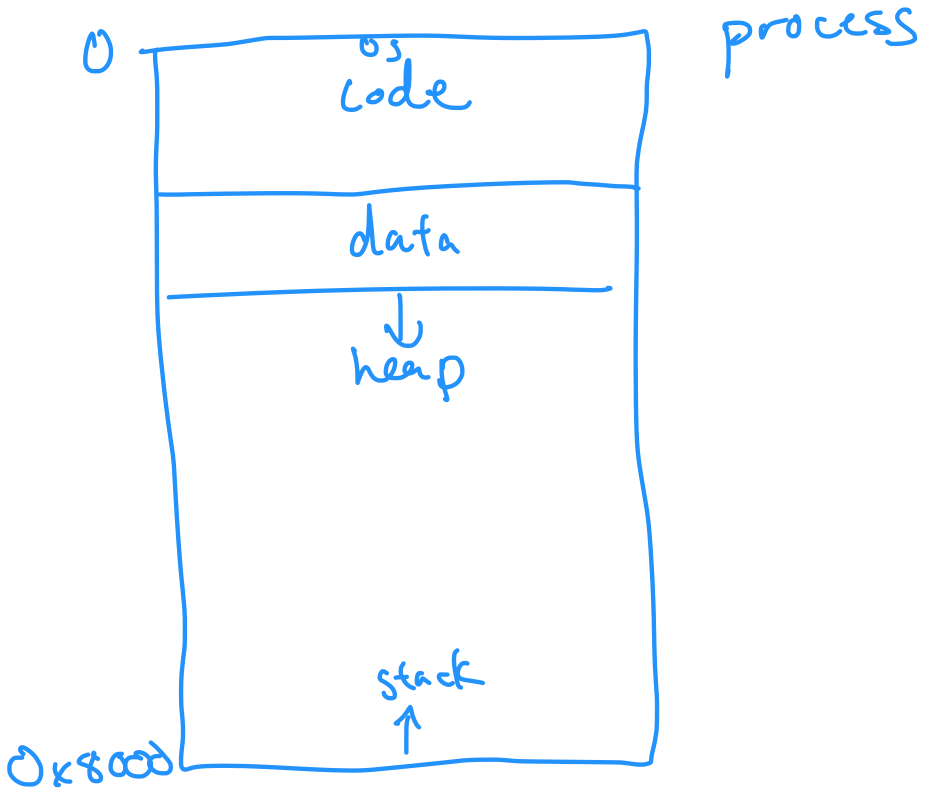
- An OS is a program that
  - manages processes
  - determines when they run
  - keeps track of what resources they're using
  - makes the computer usable



Processes

- A process is a running programming
- The OS keeps track of all the processes running a computer
    - There could be more than 1!
- Process includes all of the context of a program
- What does a process look like?

What does a process look like?



Process state — metadata

- The OS maintains the state of each process
  - PID (Program ID): a unique number for a process
  - Address space
  - Hardware resources in use (e.g., open files on disk)

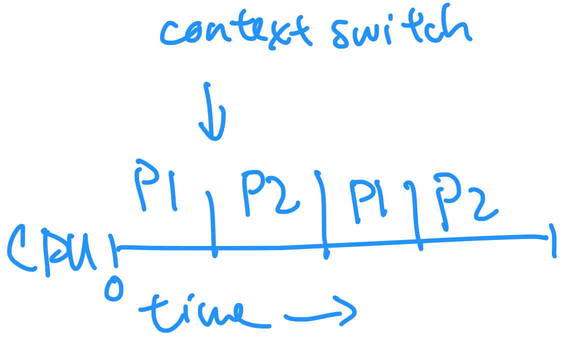
$\text{Program} \rightarrow \text{process}$

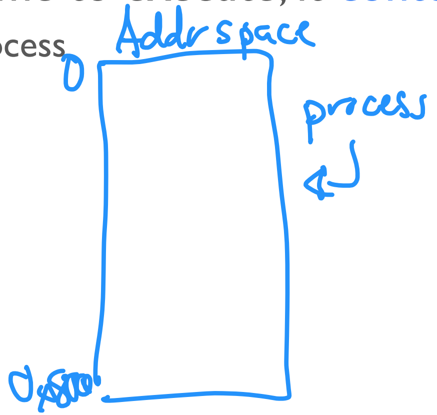
What if I want to run 2 processes at a time?

What if I want to run 2 processes at a time?

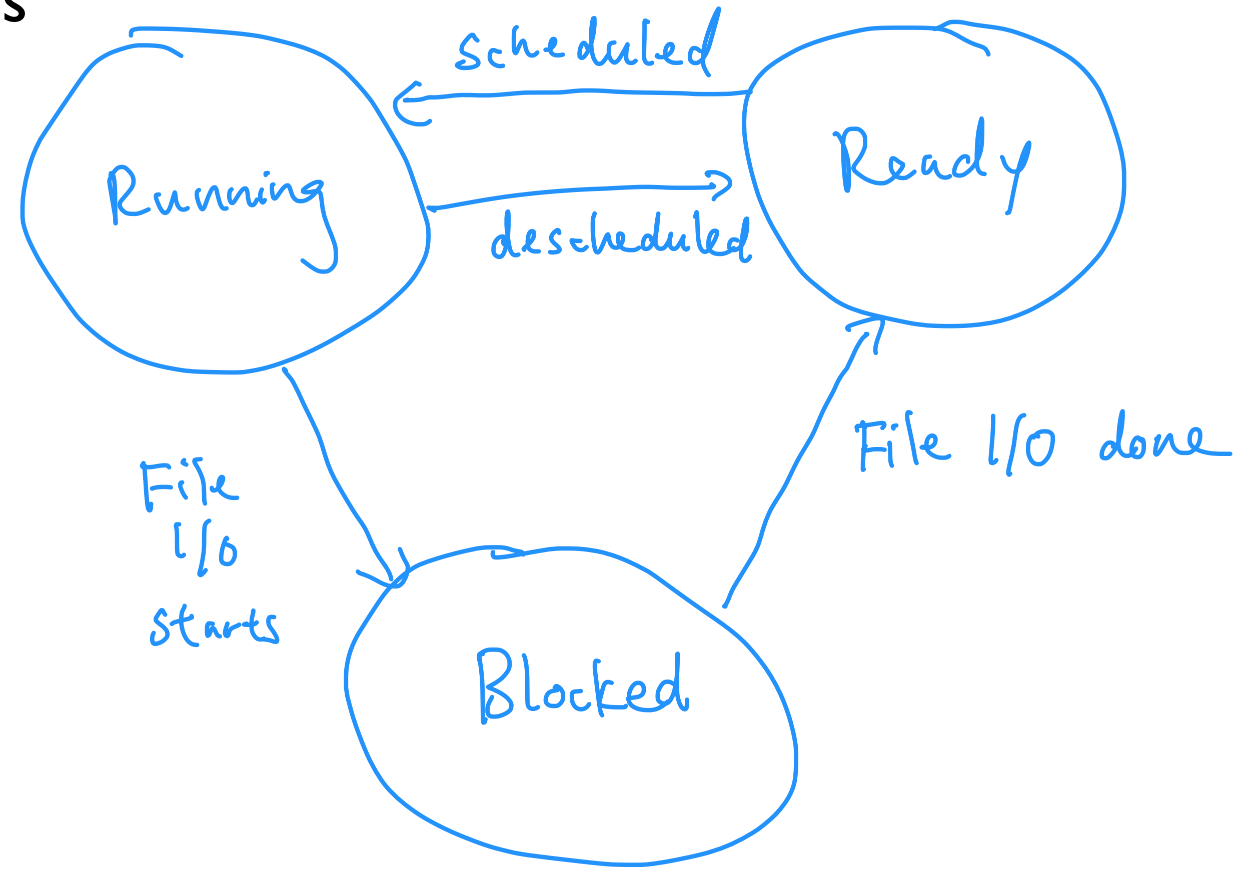
- Context switching: OS decides when to switch between processes
  - Context switching is expensive!

- When OS decides a process has had its time to execute, it context switches
  - OS saves all context/information of running process
  - New context switched process begins!





Process States



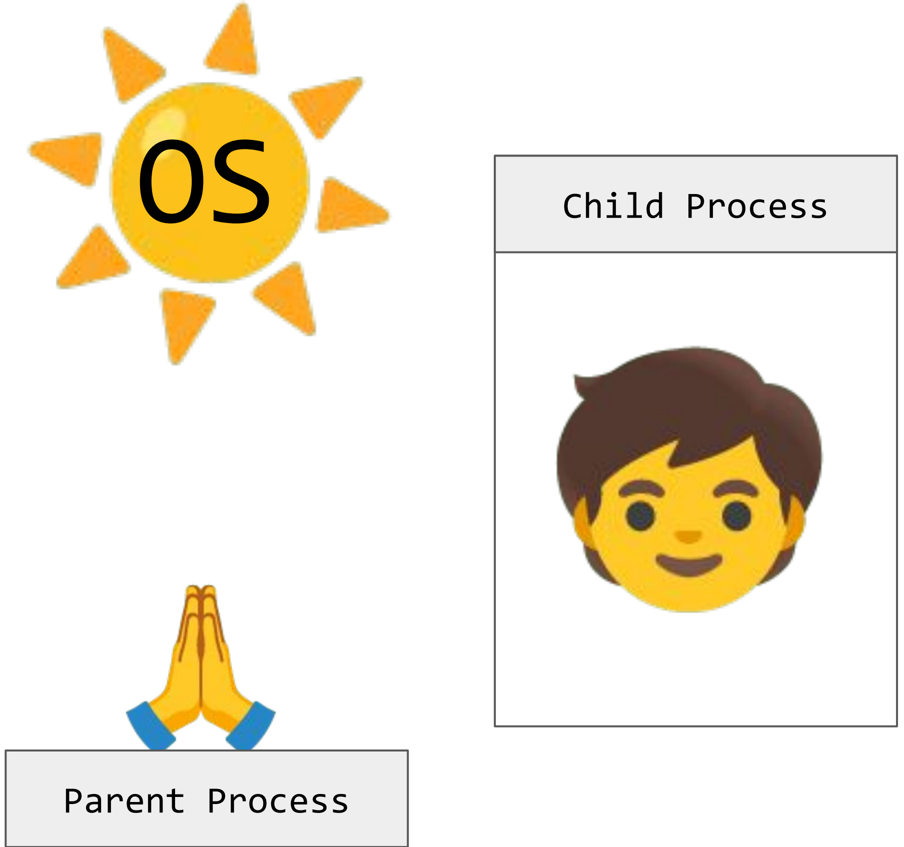
How do we create processes?

How do we create processes?

`fork()`

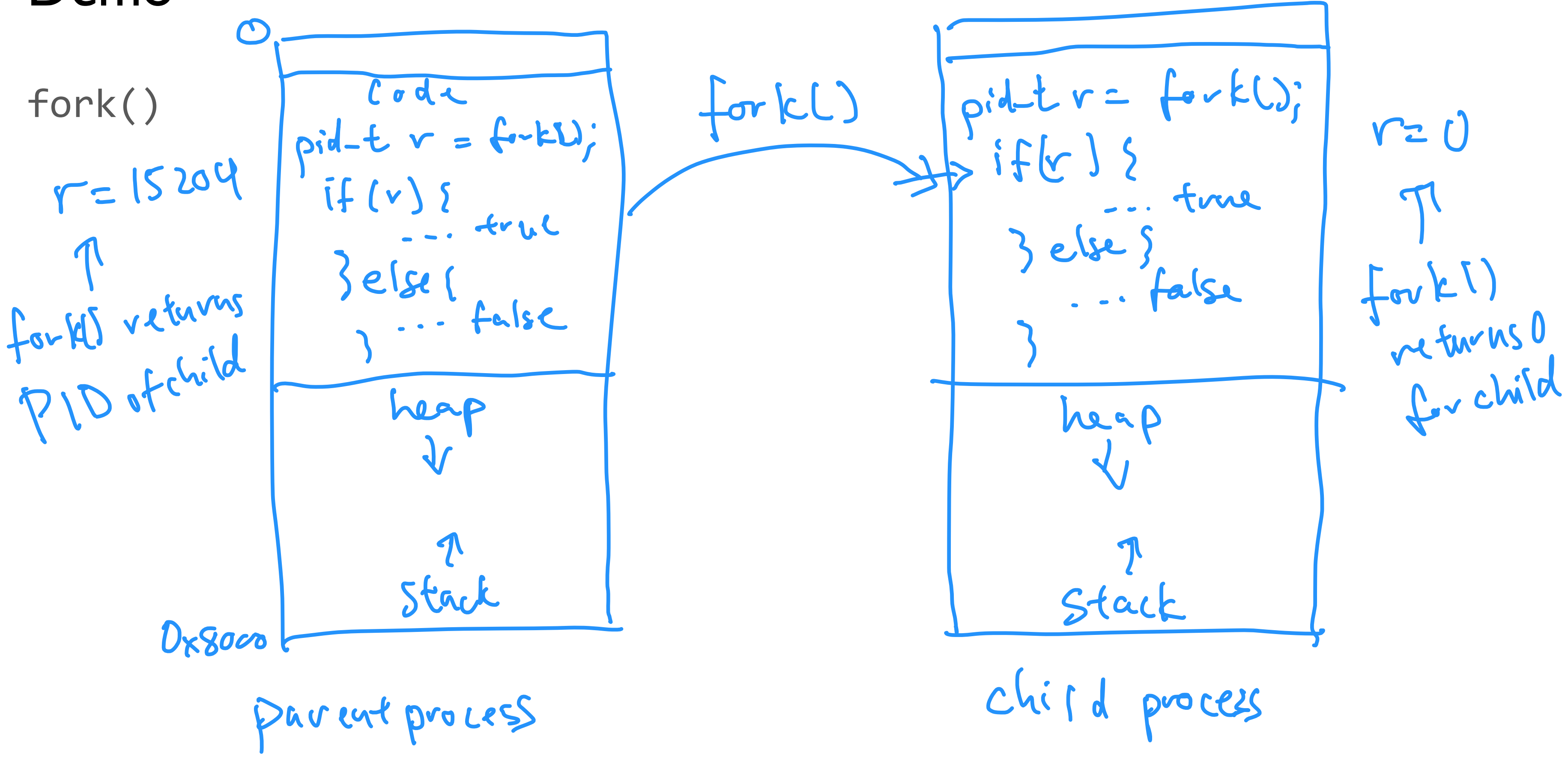
How do we create processes?

`fork()`



Demo

fork()


0
code
pid_t r = fork();
if (r) {
... true
} else {
... false
}
heap
stack
0x8000
parent process
r = 15204
fork() returns PID of child
fork()
pid_t r = fork();
if (r) {
... true
} else {
... false
}
heap
stack
child process
r = 0
fork() returns 0 for child

Which applications create and manage processes?

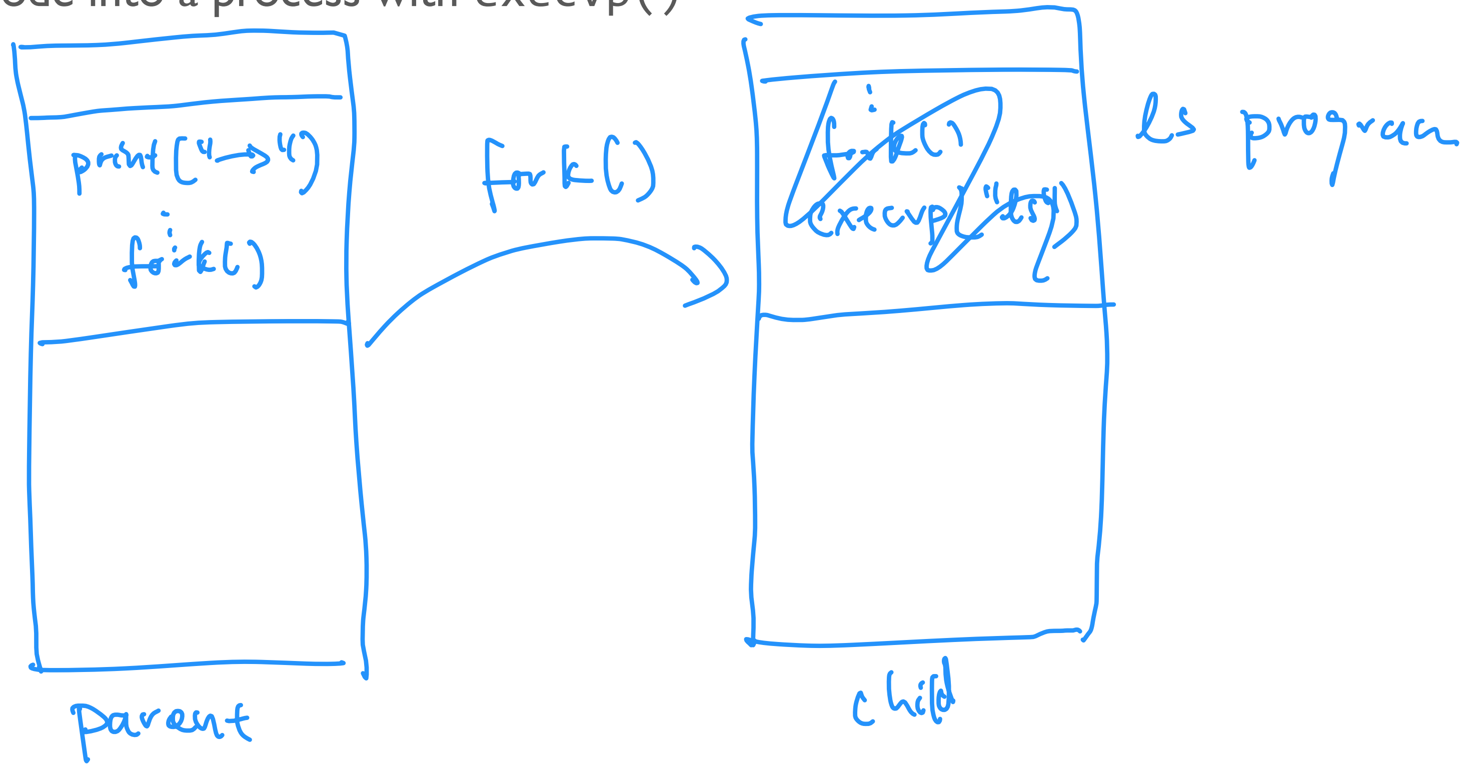
- web browser
- Youtube
- zoom
- Github

- terminal
  - ↳ zsh
  - ↳ bash

$\left.\begin{array}{l}\text{zsh}\\\text{bash}\end{array}\right\}$ shells

The shell creates, manages, and runs processes

- Loads new code into a process with execvp( )


print("→")
fork()
fork()
fork()
execvp("ls")
ls program
parent
child