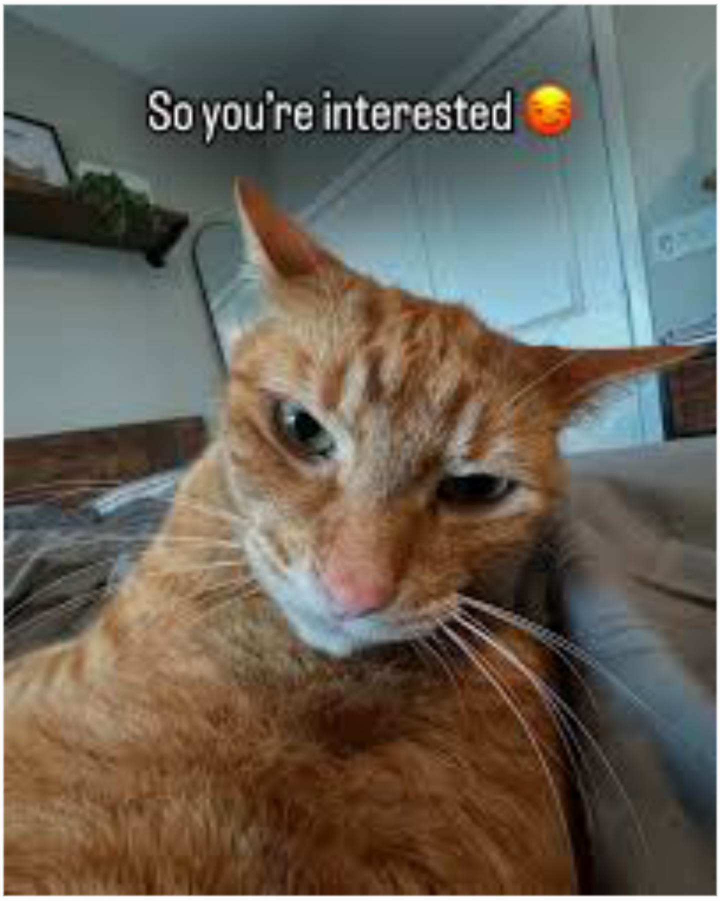
3. Process API



How to Create and Control processes

Difference between : return vs. exit, outside of main( )

- return : end that function
- exit : end that process

Fork( )

: create a new process

- copy current process

- return value : child's PID
    - x < 0 : fork failed
    - x == 0 : no child == this process is child
    - x > 0 : parent, x == child's PID

- declared line == child birth point
- Difference between Child vs. Parent : return value of fork( ), PID

- Can't guarantee which process(either child or parent) will be executed first

Wait( )

: used by a process to wait for a child process.

- return value : Child's PID
- if n children -> wait() for n time

Exec( )

: run a program that is different from the calling(parent) program. system calls that allows a child to break free from its similarity to its parent and execute an entirely new program.

- fork() : run copies of the same(parent) program.
- exec() : run a different program
- memory space (ex. heap, stack) of the program are re-initialized
- Does not create new process => using same PID
- Transform the currently running program into a different program
- successful call to exec() never returns : don't proceed lines after exec( ) in original code.

int execvp(const char *file, char *const argv[])

- vp : vector path

Isn't it possible to run a completely different program with fork()?

: Since the child and parent have different PIDs, can't we just use the return value of fork() as a branch point with if-else?

=> Technically, yes — but it’s inefficient and not recommended.

- fork( ) can work like a **branch point** like you said, with return value.
- But what about **execvp("wc", args);** ?
- exec() is like a **magic spell** — it replaces the current process with a new, independent one. It's for true execution.

- In theory, you could write the logic of wc directly in the child branch. But it would be: redundant, hard to maintain, completely non-portable
- exec() is more portable, modular, clean way.

- **If fork() gives you a two-way branch like a fork in the road, exec() is Doraemon's anywhere door — it teleports the process into a different world (program).**

Separation of fork() and Exec()

- run code after the call fork() ~ but before the call exec();
    - use for environment setting

1. User : type a command
2. Shell : figures out where in the file system the executable resides
3. Shell : calls fork() to create a new child process to run the command
4. Shell : calls exec() to run the command
5. Shell : waits for the command to complete by calling wait()
6. Shell : When the child completes, the shell returns from wait()
7. Shell : prints out a prompt again, ready for your next command

File descriptors assignment in UNIX

- UNIX systems start looking for free file descriptors at 0.
- It's important to close right descriptor -> make empty and use by open()
- 0 : STDIN, 1 : STDOUT, 2: STDERR

- Close : empty that slot
- Open : fill that empty slot

Process Control And Users

: for cummunication, a process shoud use the **signal()** system call to "**catch**" various signals : stop, continue, terminate

- Users can only control their own processes
- It is the job of OS to parcel out resources to each user (and their processes) to meet overall system goals