


⑤ Interlude: The Process API



- * Discuss process creation in Unix systems

CPUX: How to create & control processes?
↳ functional, easy to use, & high performance.

5.1 The fork() System Call

- * fork() system call is used to create a new process.

↳ watch out for fork bombs!

$$\text{rc} = \begin{cases} < 0 & \Rightarrow \text{failure} \\ 0 & \Rightarrow \text{child} \\ > 0 & \Rightarrow \text{parent (pid of child process)} \end{cases}$$

- * child of fork is a process that is almost an exact copy of the process that invoked fork() (i.e., parent) — own private address space, registers, PC ...
- * child starts running where fork was invoked.
- * outcome is not deterministic as depends on scheduler.
↳ leads to problems w/ concurrency.

5.2 The wait() System Call

- * often useful for the parent to wait for a child process to finish executing. (wait())
↳ waitpid()
↳ use can make code deterministic

5.3 The exec() System Call

- * exec() is useful when you want to run a different program from the calling program.
↳ loads code/static data from the executable you want to run & overwrites the current process' memory space (heap/stack/static data).
↳ i.e. — transforms currently running program into another running program.

5.4 Why? (API)

* `fork()` & `exec()` is essential in building a UNIX shell.

↳ executing command := `fork()` & `exec()` w/ command & args,
return to parent process for next command.

↳ Output redirection := `fork()`, close `stdout` but open file, `exec()`

↳ OS looks for file descriptors @ zero to output ~~for~~ (`STDOUT_FILENO`)

↳ essentially reassigning `stdout` to a file. (instead of screen)

↳ pipes := use `pipe()` system call

↳ output of one process connected to in-kernel pipe (queue)

↳ input of one process connected to same pipe

5.5 Process Control & Usage

* `kill()` = syscall used to send signals to a process (pause, die, etc.)

* `SIGINT` - interrupt (terminate)

* `SIGSTOP` - stop (pause)

* signal subsystem allows for sending signals to whole process groups

* process uses the `signal()` syscall to catch various signals.

↳ restrict access to what processes can catch signals

↳ user can exercise control over the processes they launch

root user can kill arbitrary processes they didn't launch, shutdown the system, etc.

5.6 Useful Tools

* `ps` command - info on processes

* `htop`

* `kill`

(5.7) SUMMARY

* spawn() > fork() ?