CS 219 LECTURE 7

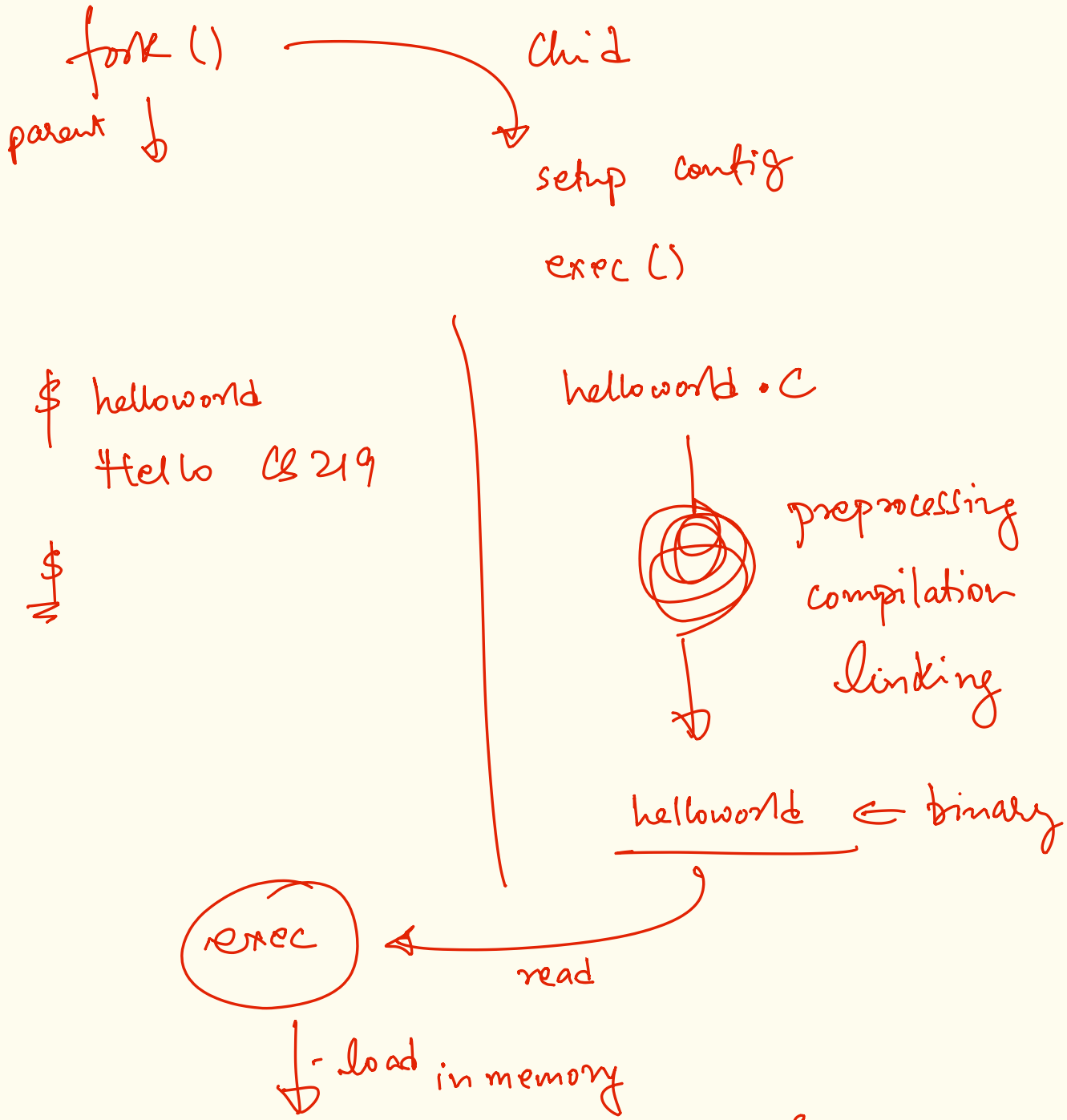
the process API (for engaging/using the process abstraction)

the system call mechanism

~ process system calls —

fork, exec, wait, open, close, read, write, getpid, getppid, sleep

- provides a mechanism to create new instances of execution.
- can be useful for instances of parallelizing work
  - matmul
  - concurrent web requests
  - ...
- combined with exec
  - setup/load new programs & attach to a process.
- fork + exec
  - fork: setup process config.
    - fds, cwd, ...
  - exec: program to load.

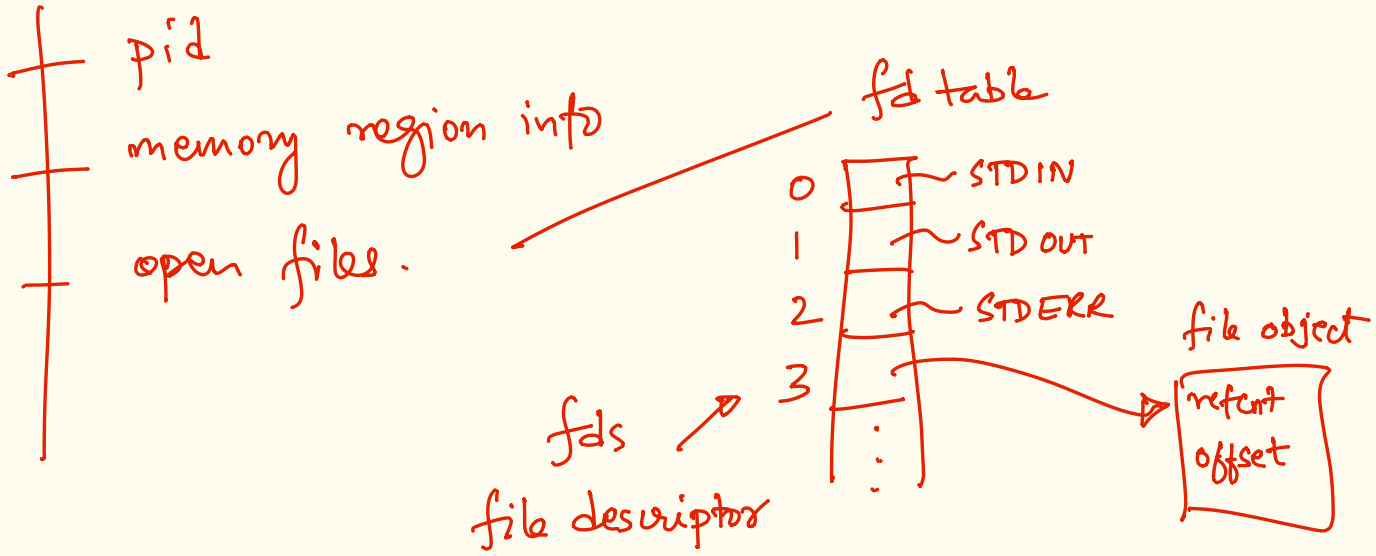


e.g.

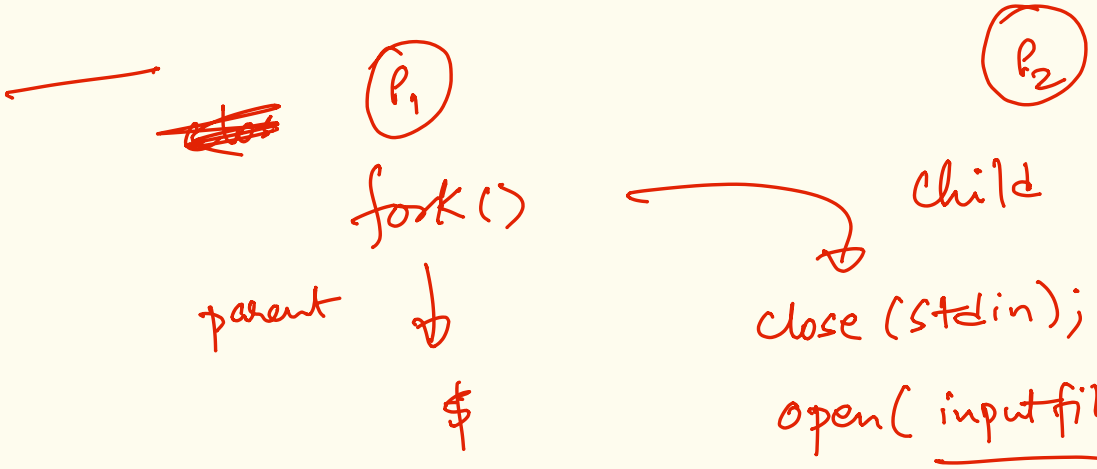
```
$ cat
hello
hello
^C
$ cat helloworld.c
#include <stdio.h>
   :
$
```

```
cat.c
|
while (1) {        fd 0
  read(stdin)
  write(stdout)
                   fd 1
}
```

PCB – process control block



open systemcall chooses the lowest fd number that is free.



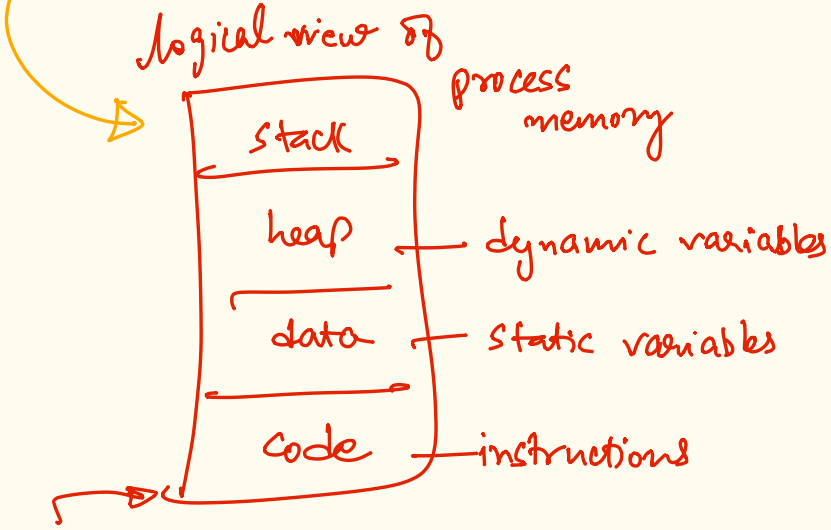
- ls | wc -l
- cat > input.txt
- ps -aux | grep hello > output.txt

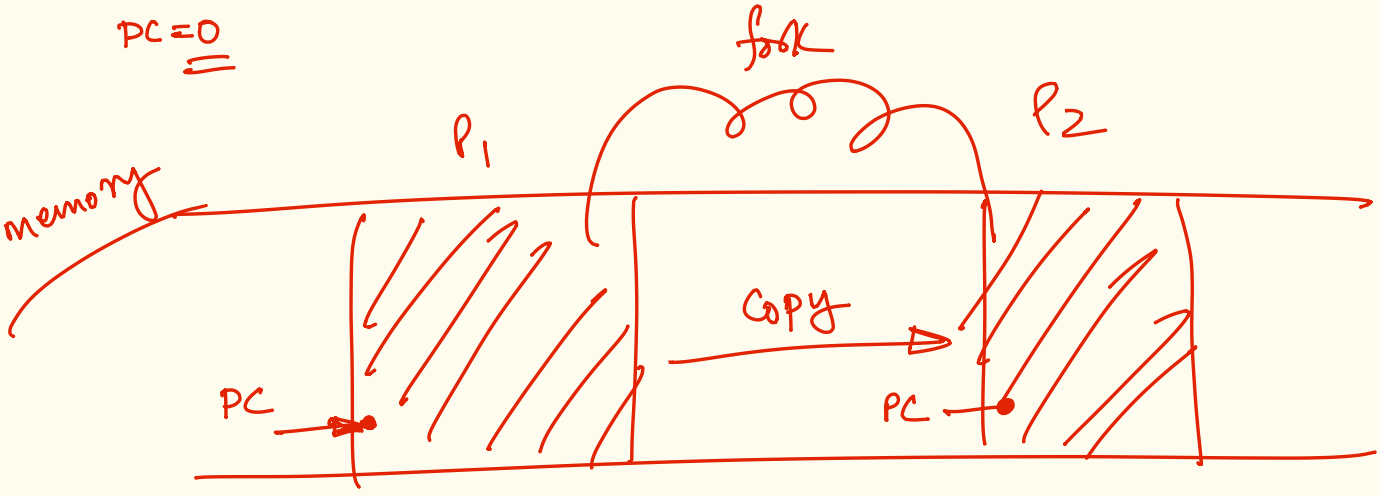
fork → copies all process state

- duplicates
  - create a new PCB
    - copy contents of parent PCB to child PCB
    - copy all memory contents of parent process

metadata

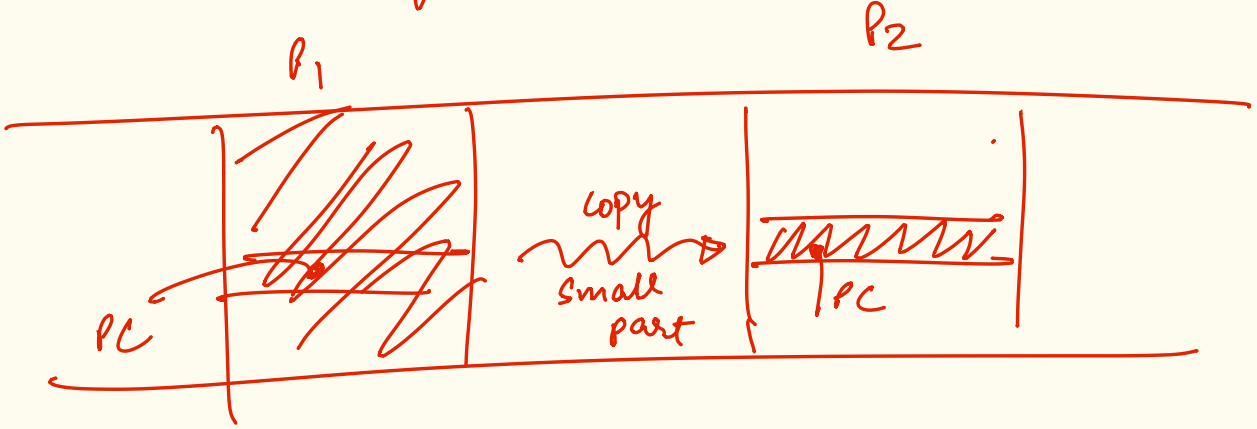
execution state.





(#) copy of process state is an overhead if fork followed by exec.

CoW ~ copy-on-write optimization.



(?) how do know when state/memory is needed?

- protection memory regions via R/w access bits.

~ turn-off R/w access. & child process

~ and write-access in parent process.

~ on R or W access in child — exception!

or write access in parent

exception "handled" by the OS.

copy required / appropriate memory state! from parent to child process.