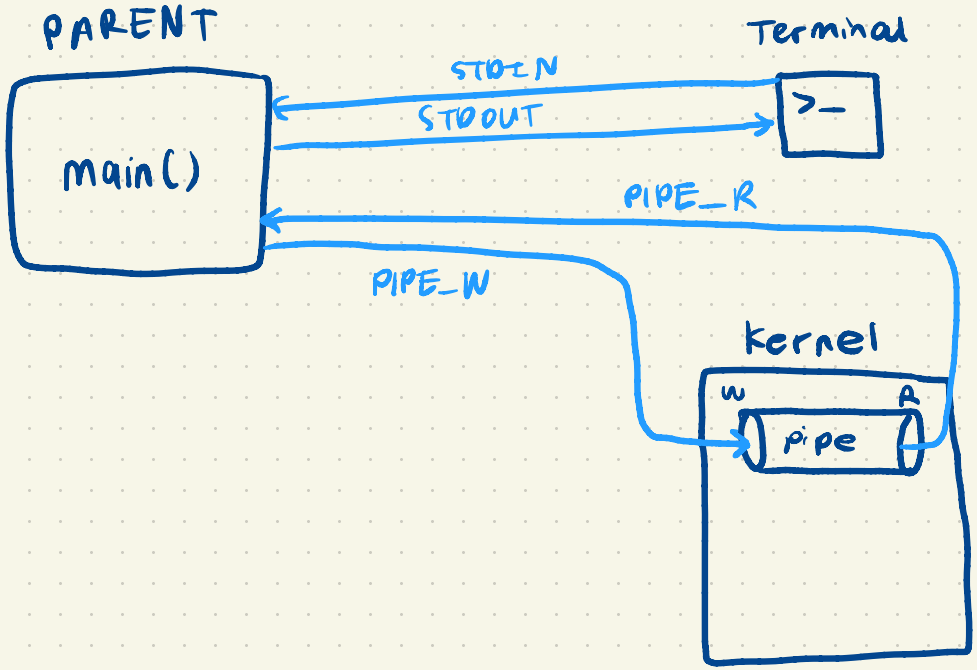
user input : ` ls | wc `

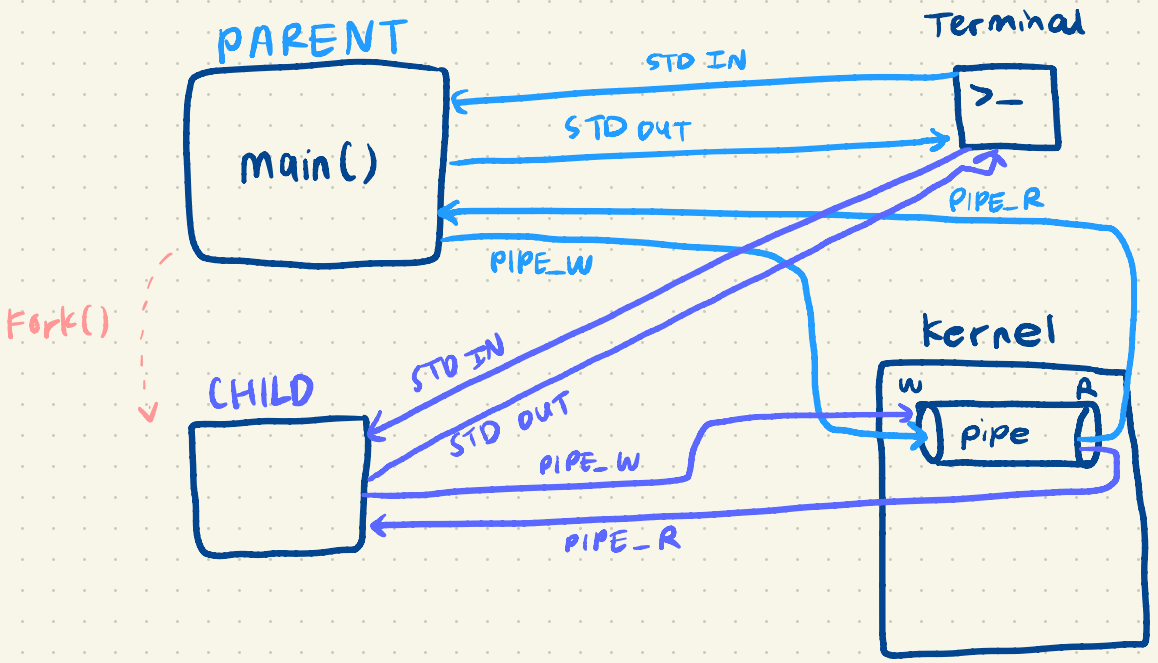
user input has 2 commands and 1 pipe.

∴ 1 pipe buffer is created
& fork() is called twice

PARENT calls pipe.



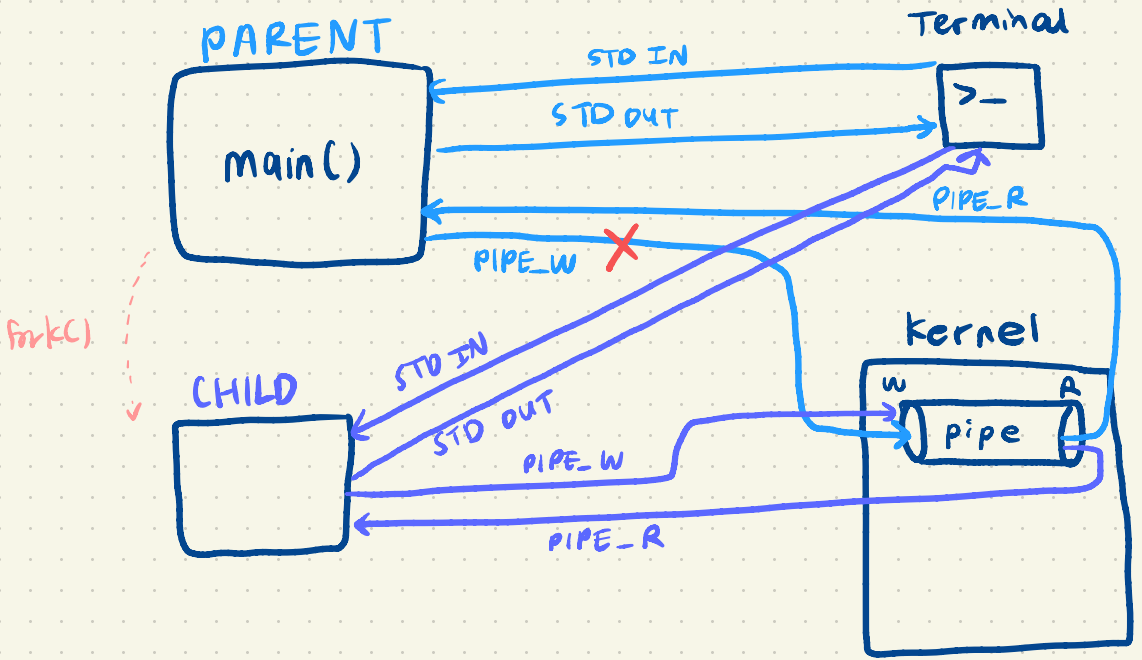
PARENT – calls fork(), which spawns a child process. This child will run the `ls` cmd. child has access to an independent copy of parent's file descriptor table.



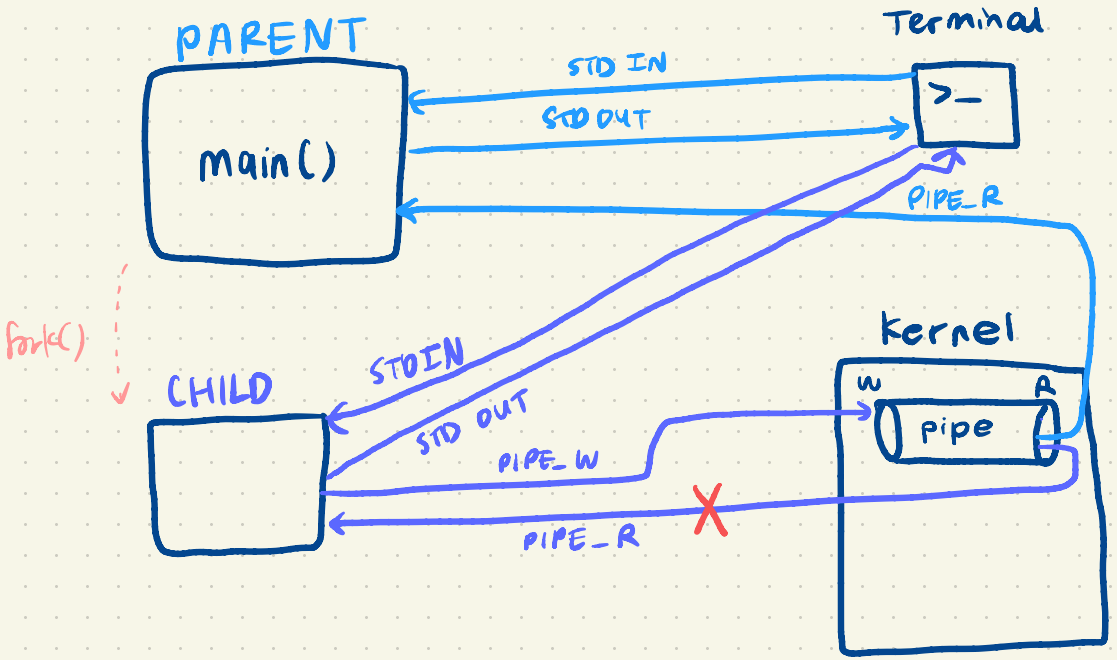
clean up file descriptors.

ls | wc

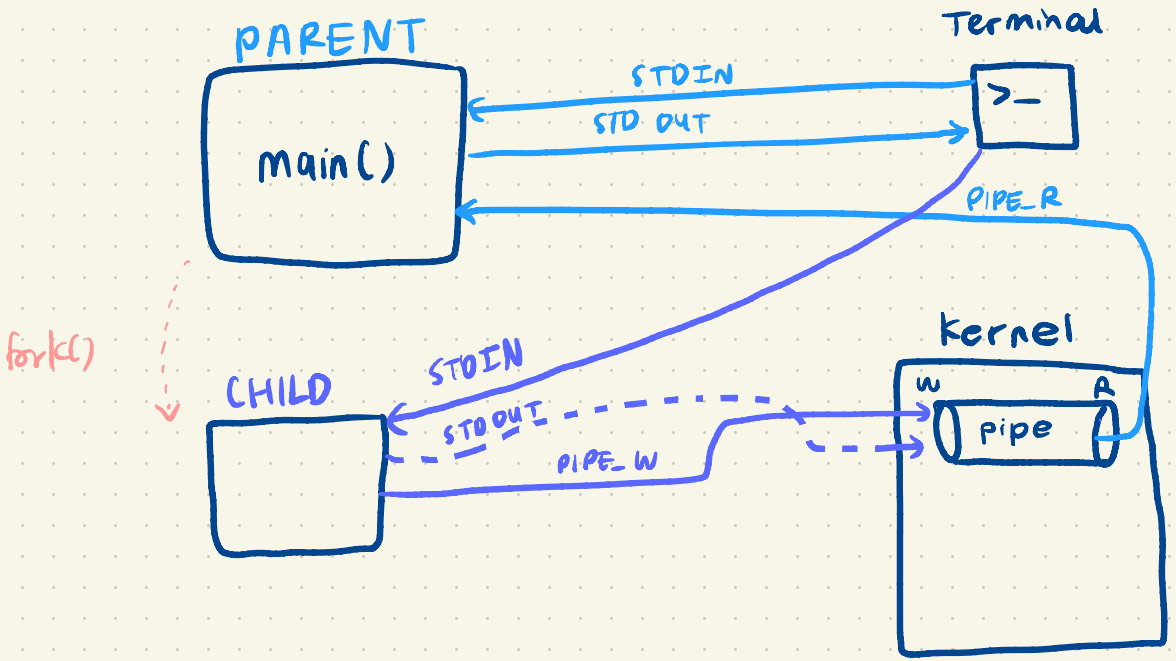
PARENT - Does not need write access to pipe - close pipe_w.



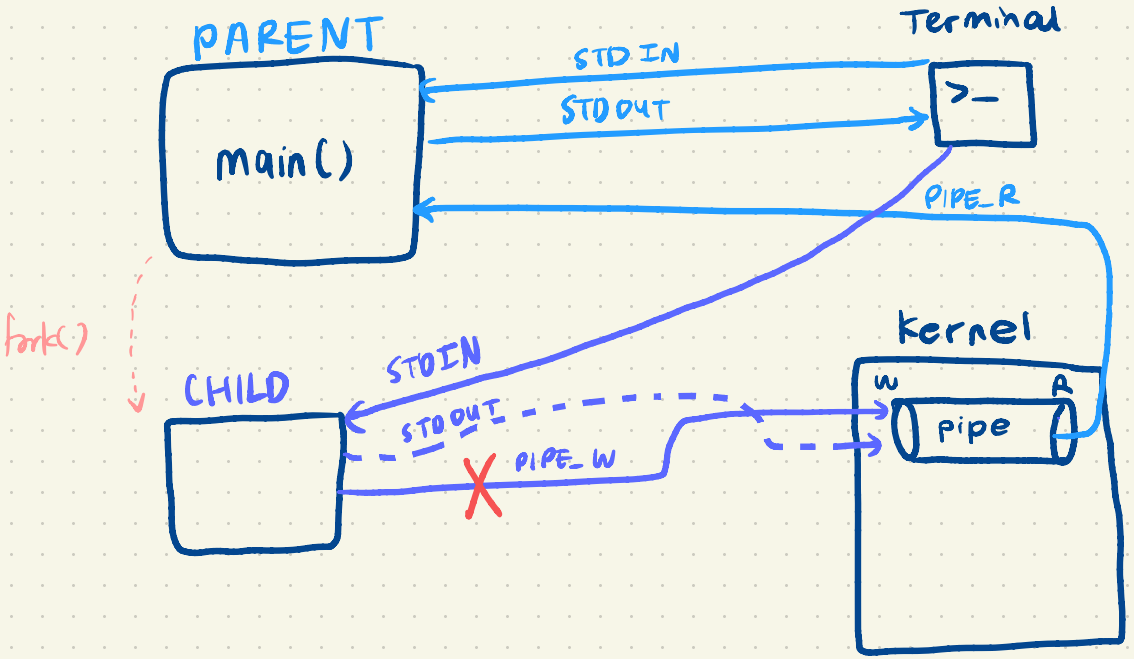
CHILD does not need to read from pipe.
close PIPE_R



CHILD needs to write to pipe (so that the next command can read from it)
Redirect CHILD's STDOUT so that it writes to pipe (instead of terminal).

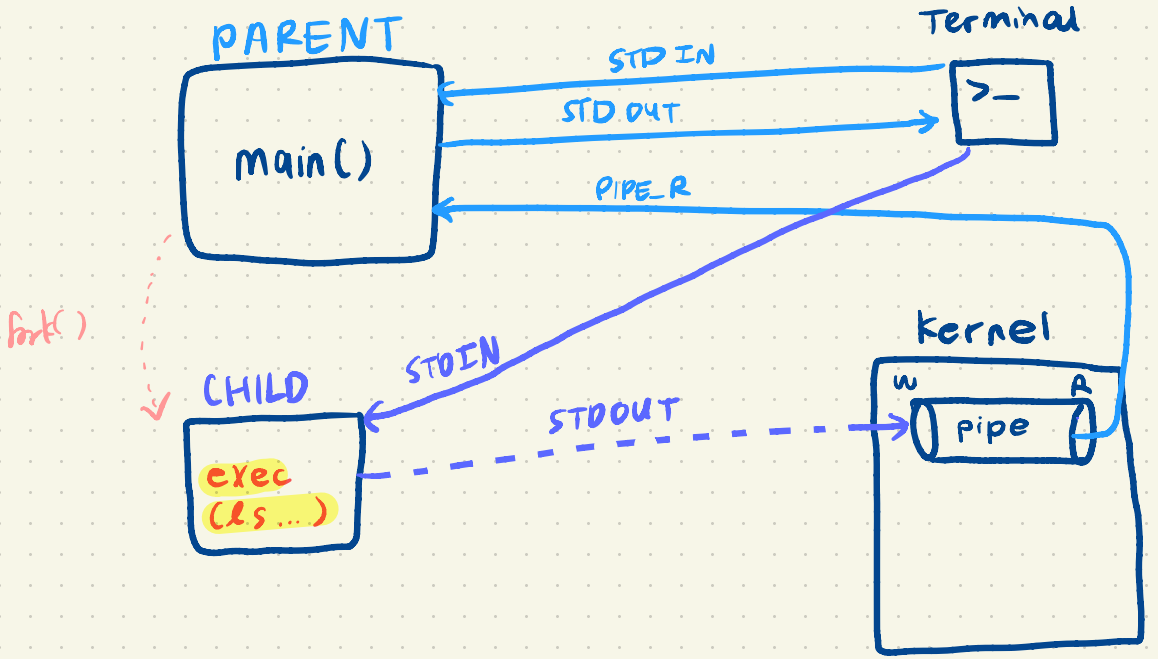


Now, CHILD has 2 write access points to pipe. Close pipe_w.

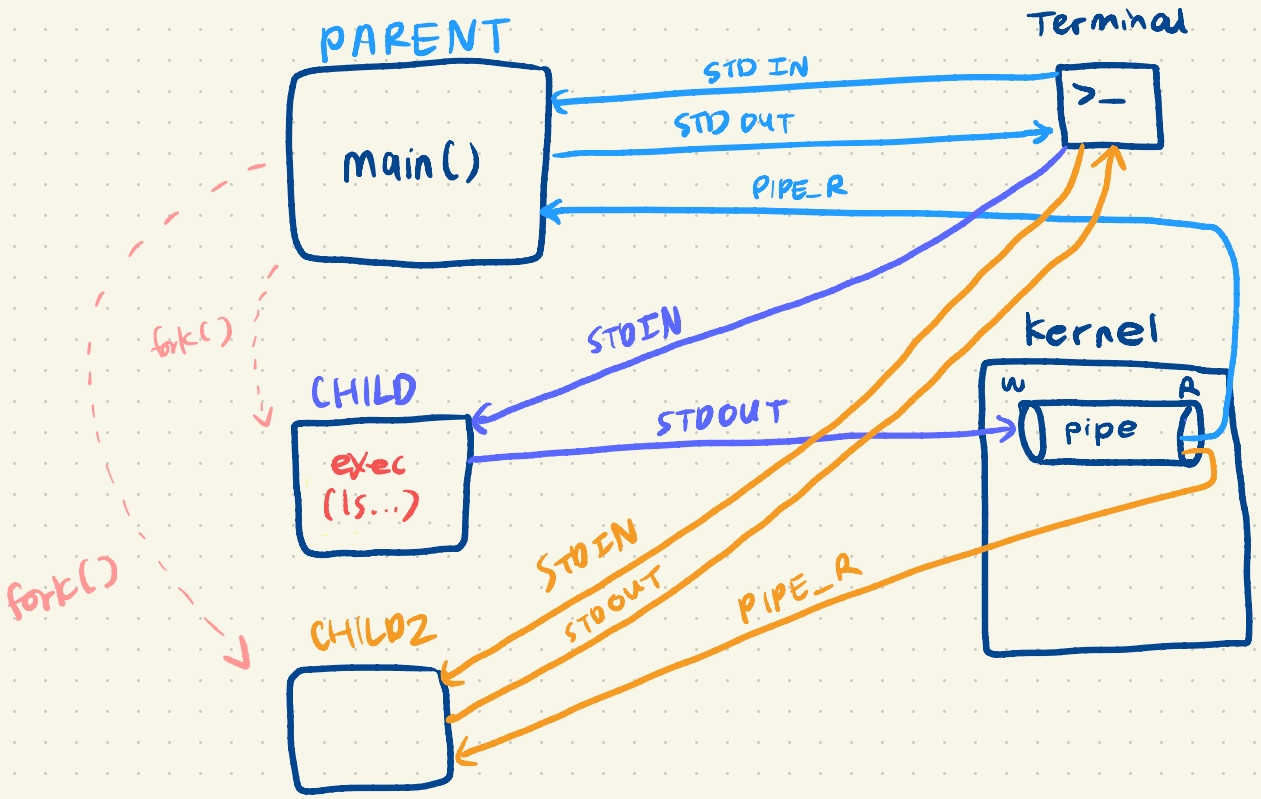


CHILD now reads from Terminal and writes output to pipe.

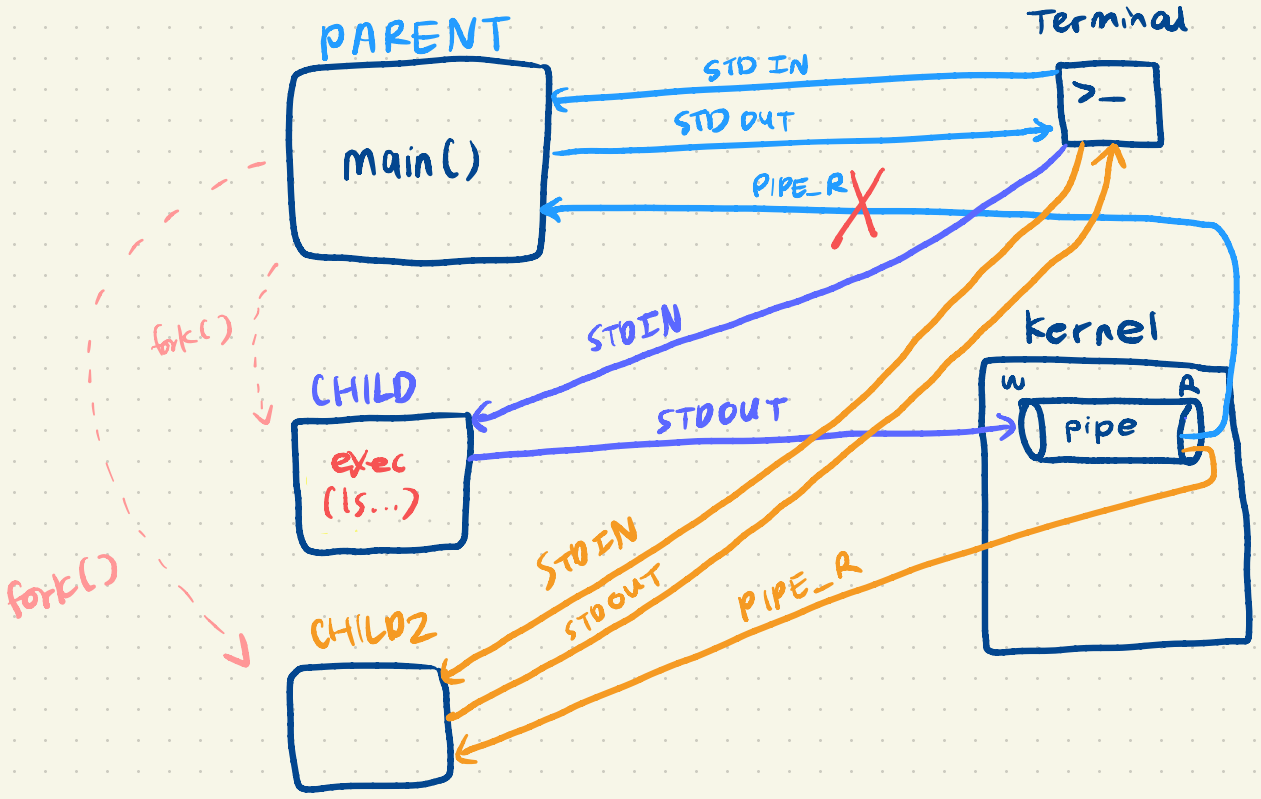
CHILD can now call `exec("ls", ["ls, nullptr])`



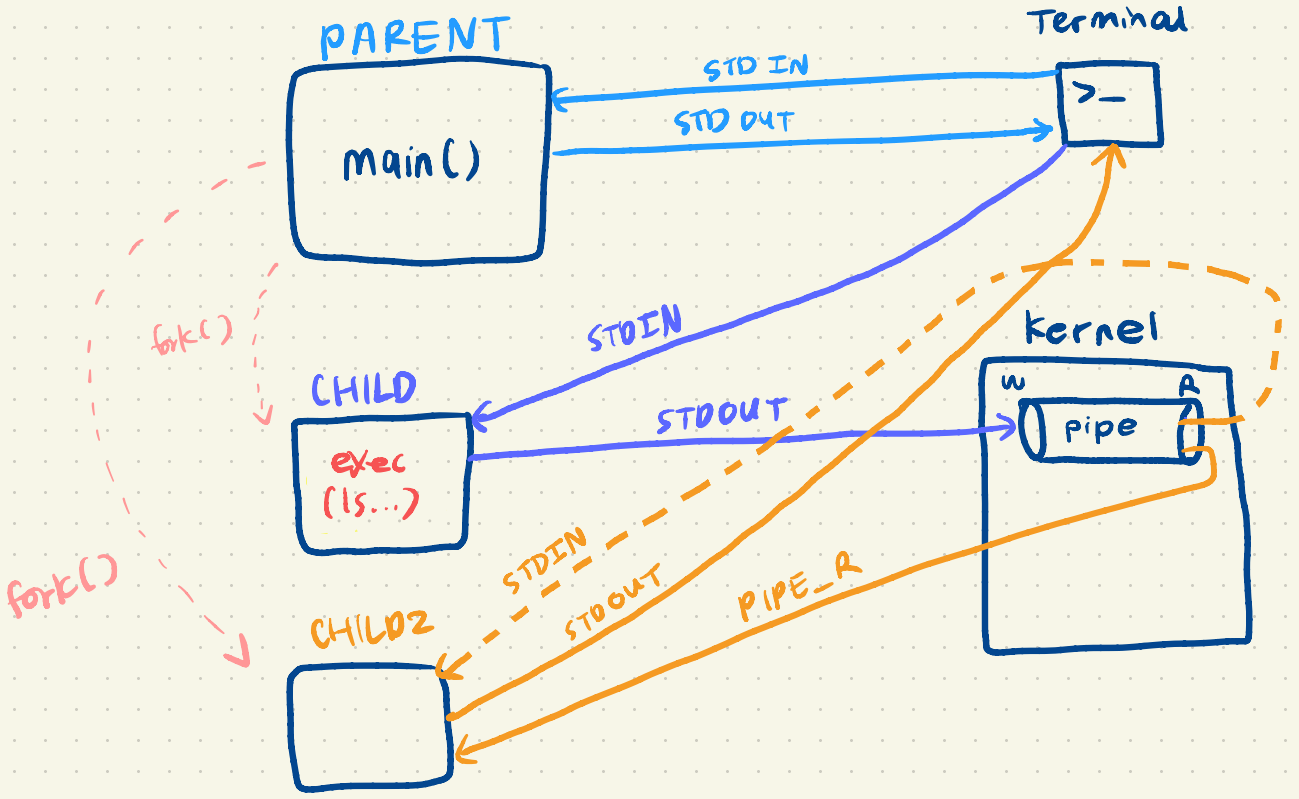
PARENT can now call fork() again to spawn the process that will handle the command `wc`. CHILD2 has the same access as parent:



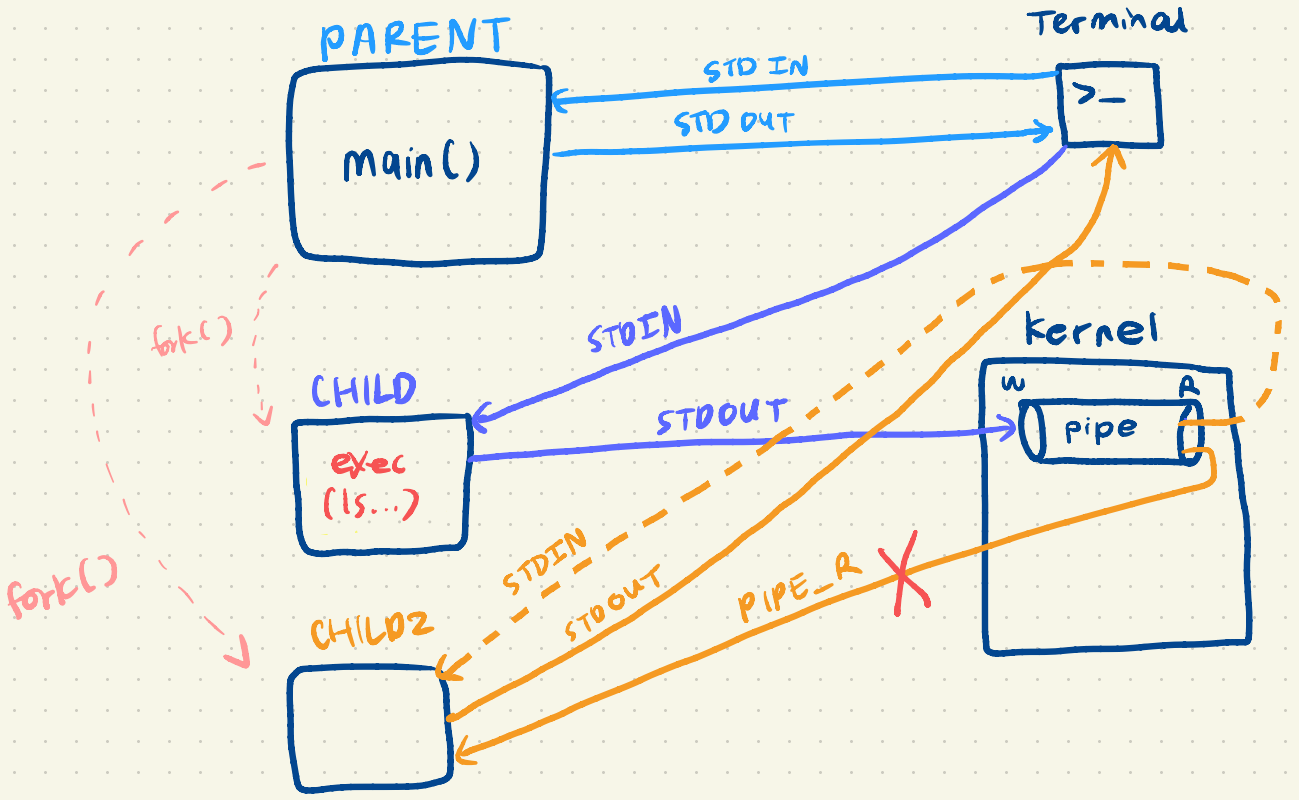
PARENT does not need Read access to pipe anymore. Child2 has been spawned (and it's the last command). Close pipe_R.



CHILD2 needs to read from pipe (output of `ls` will be in pipe).
So it will REDIRECT STDIN to pipe.

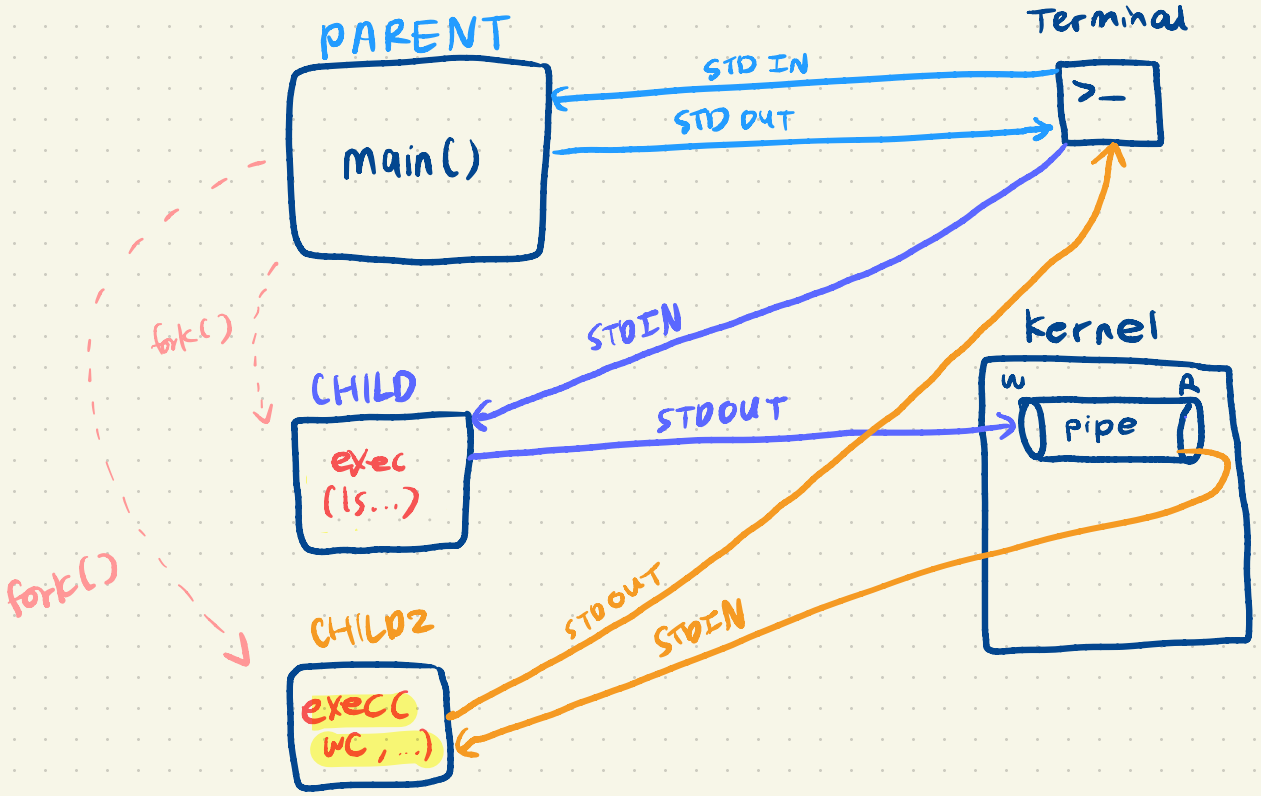


CHILD2 now has 2 read access points from pipe. Close pipe_R.

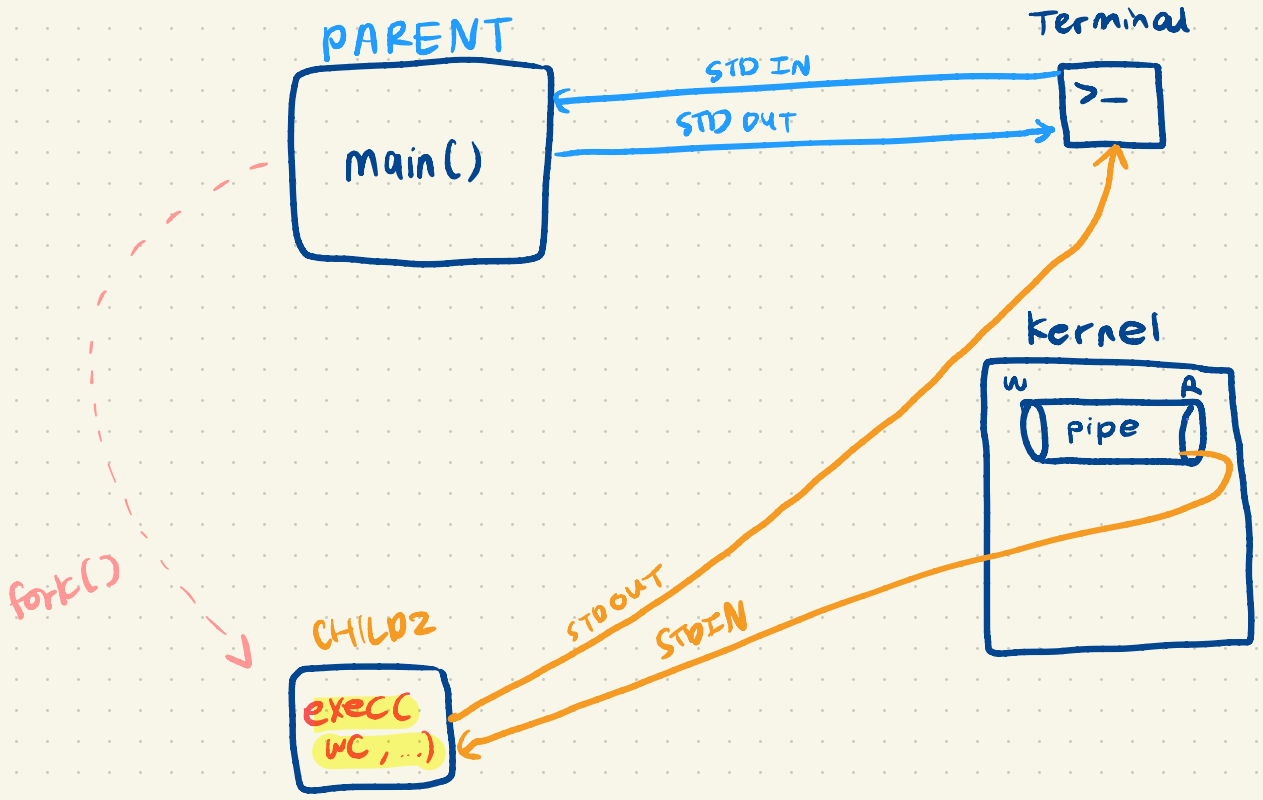


Now CHILD2 reads from pipe and writes to terminal.

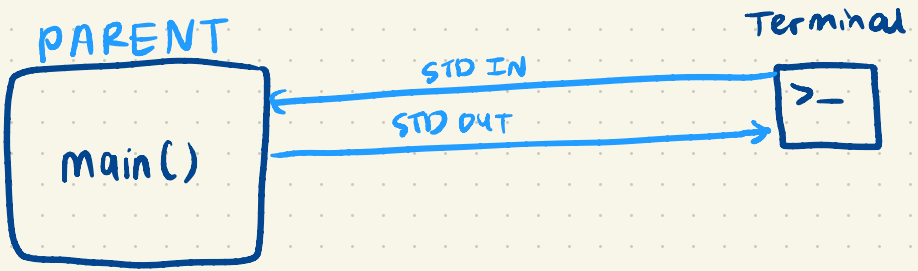
CHILD 2 can call `execC("wc", ["wc", nullptr])`

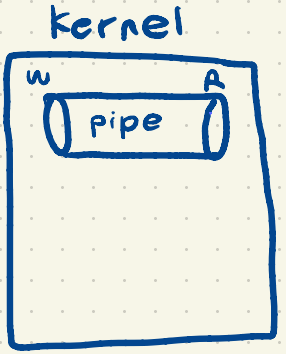


PARENT will call waitpid. When CHILD finishes running, it sends the output of 'ls' to pipe and terminates.



CHILD2 reads from pipe and executes `wc`. If child2 tries to read from pipe and there's nothing there AND no one can write to pipe, it reads EOF and terminates.





PARENT prompts user for next command.