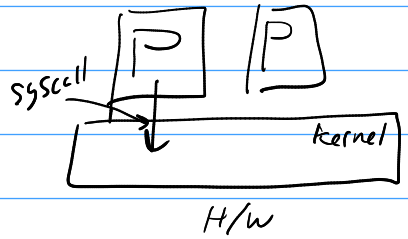
CS 3214 Lecture #6 "file descriptors & pipes"

Processes: abstraction of running program
- dual mode / limited direct exec.
- context switch
- process states
- creating new processes

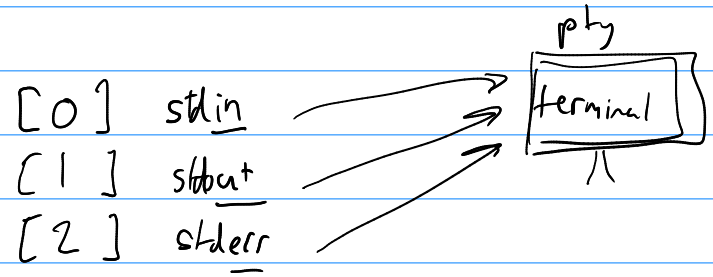


fork - new process
exec - new program

new process, old prog
clone of (heap, stack, fds, ...)
called 1x returns 2x

new program, same proc.
reinit heap, stack, ...
keep fds
called 1x no returns

File descriptors



int fd = open (..., ...

"handle" to abstract kernel obj
- terminal
- socket
- pipe IPC
- file "seq. of bytes"

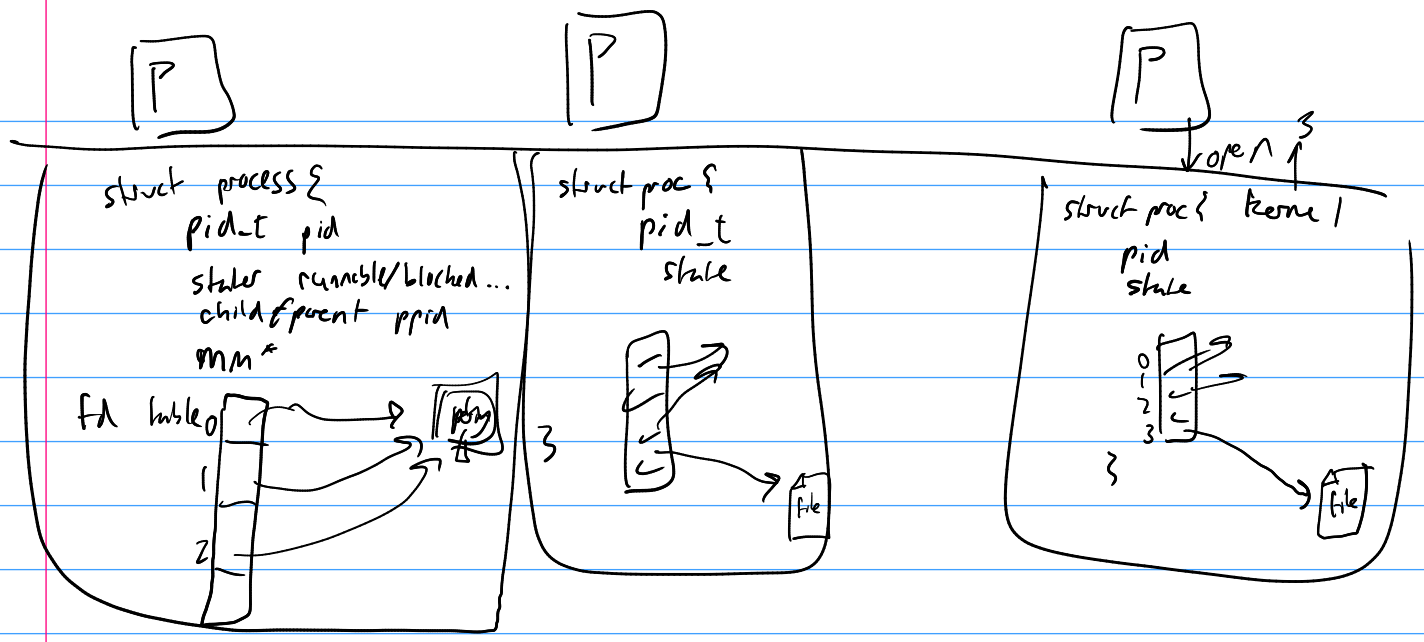
UNIX: unified interface

plan 9

fd = open
read (fd, ...
write (fd, ...
close (fd)

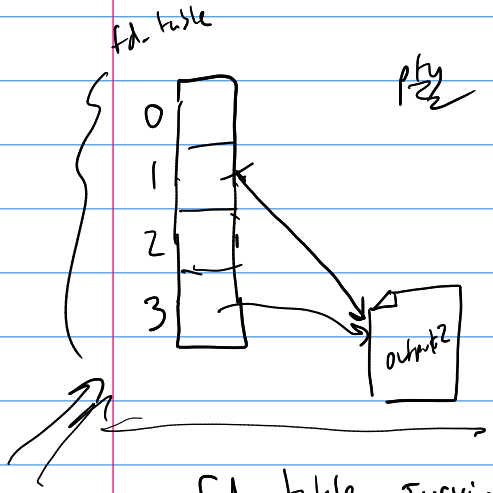
file descriptor
anything
user space handle

{ lseek
dup2 (rename/rewire)



redirecting file descriptors cat foo > output

stdout → file



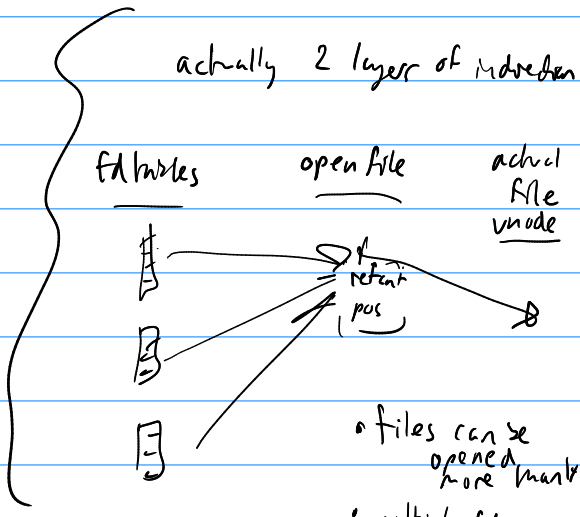
fd = open ("output 2")

3 ←

dup2 (fd, 1)

3 → 1

fd table survives exec!



- files can be opened more than once
- multiple fds point to same file

close (fd): decrement refcnt in open file

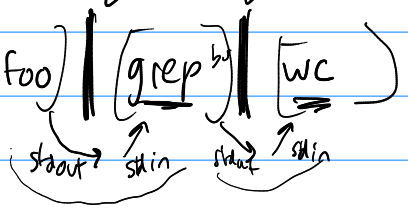
automatic close of all fds on process exit

Is redirection all that important? YES

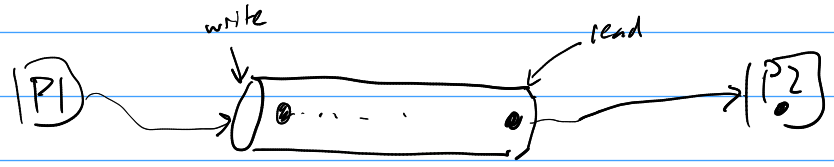
PIPES!

Pipes: cat foo | grep bar | wc

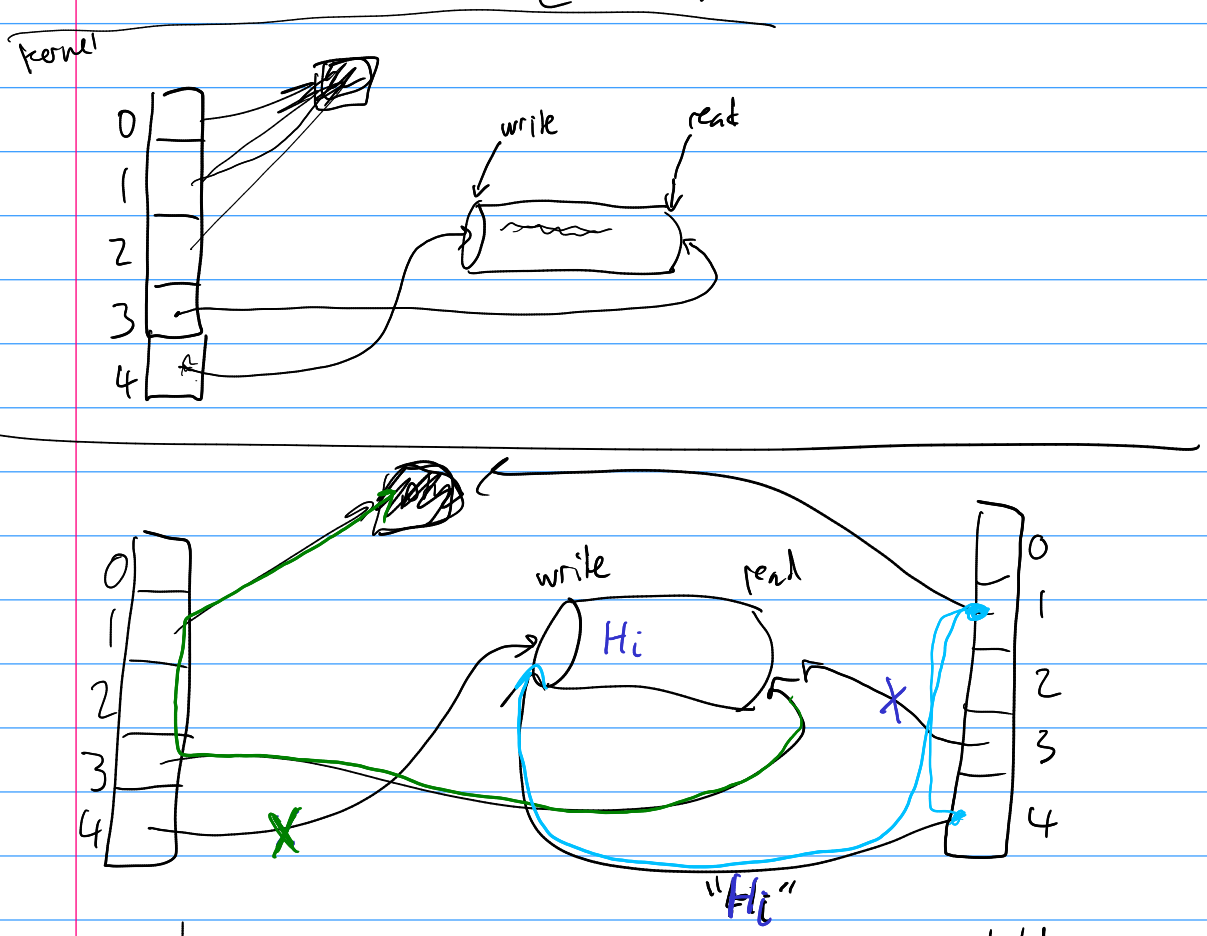
shell pipe



more generally: kernel managed way to communicate b/w processes



bounded buffer
Synchronization
reader too fast
writer too fast



fork preserved fd's
exec preserved fd's