# Processes & IPC Applied

Lecture 25
Class 27 of 28 | April 25th 2023 | COMP 211-002 | Joshua Bakita

## Front Matter

Welcome!

Today:
- ➔ Processes & IPC, Resumed
- ➔ Exam Review Information

Logistics:
- ➔ For regrade rqs., prefer Gradescope or Pizza
- ➔ Research opportunity if you get an A/A-

Fun fact…

*Sherly is currently the #1 leader on the Assignment 5 hacked save leaderboard.*

## Inter-Process Communication (IPC) and Process

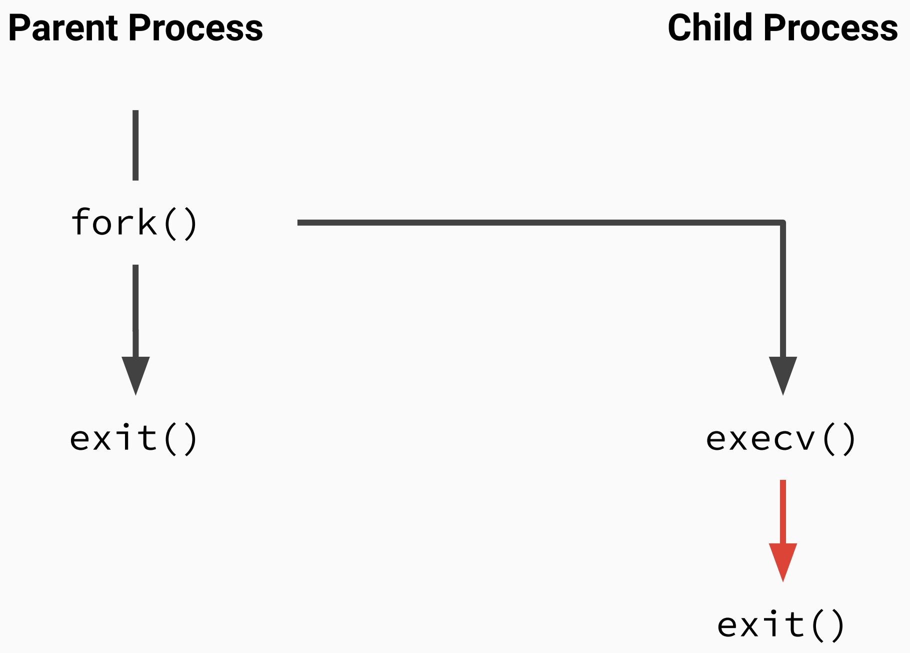
Very relevant to Assignment 5!

Building off the code from last time:
https://cs.unc.edu/~jbakita/teach/comp211-s23/l24/class_demo.c

## Last time...

We:

- Used fork() to duplicate our process
- Used execv() to replace the new (child) process with an instance of modify



IPC & Processes

## Which `exec` function to use?

Which one to use?

`execv(char* pathname, char* argv[])`

See `man execve` for details. `execv()` simply automatically passes the environment:

> *"All other exec() functions (which do not include 'e' in the suffix) take the environment for the new process image from the external variable environ in the calling process."* (`man execv`)

IPC & Processes

## What arguments does `exec` take?

```
execv(char* pathname, char* argv[])
```

`pathname`

*"All exec() functions (which do not include 'p' in the suffix) take as their first argument a (relative or absolute) pathname that identifies the program to be executed."* (`man execv`)

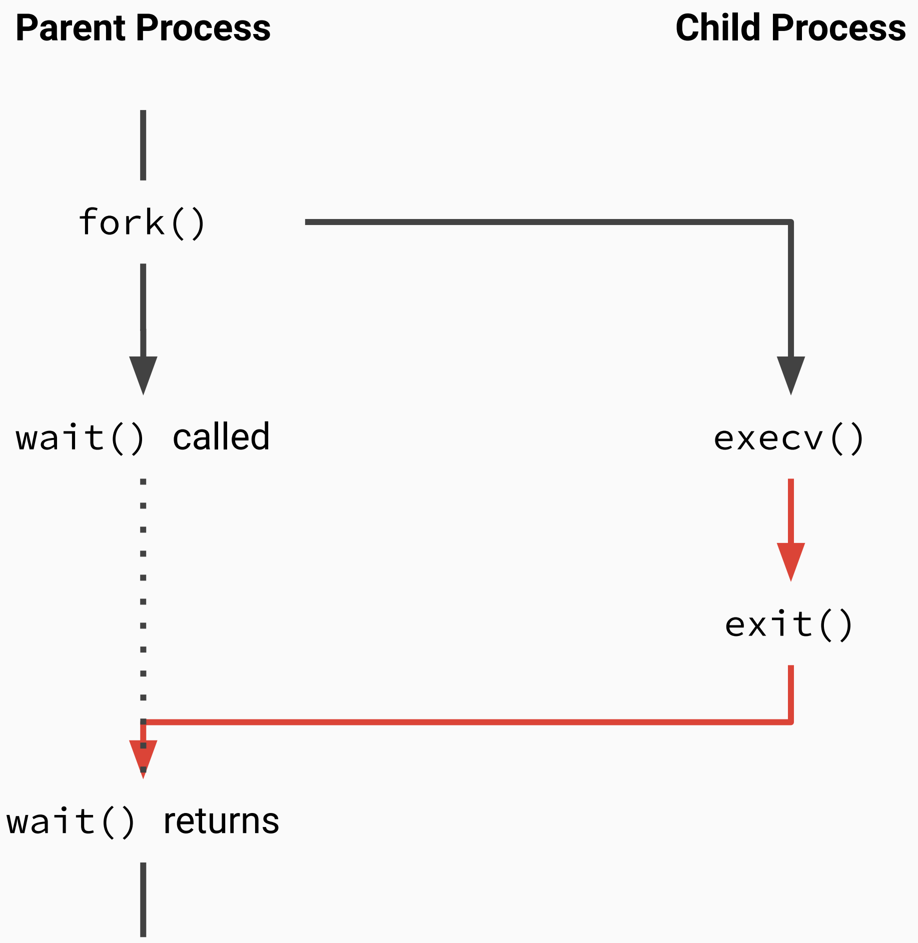
`argv`

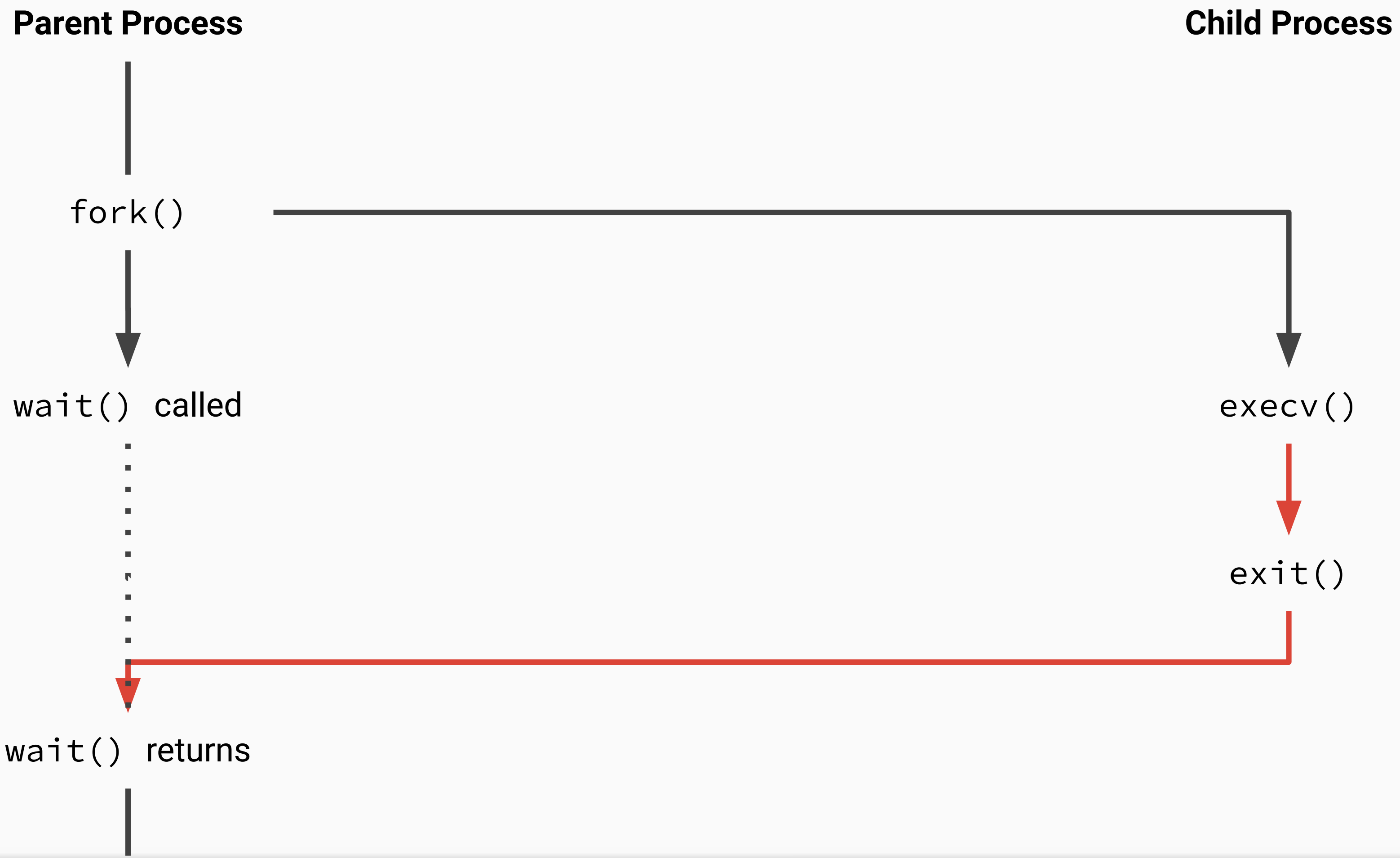
*"The char *const argv[] argument is an array of pointers to null-terminated strings that represent the argument list available to the new program. The first argument, by convention, should point to the filename associated with the file being executed. The array of pointers must be terminated by a null pointer."* (`man execv`)

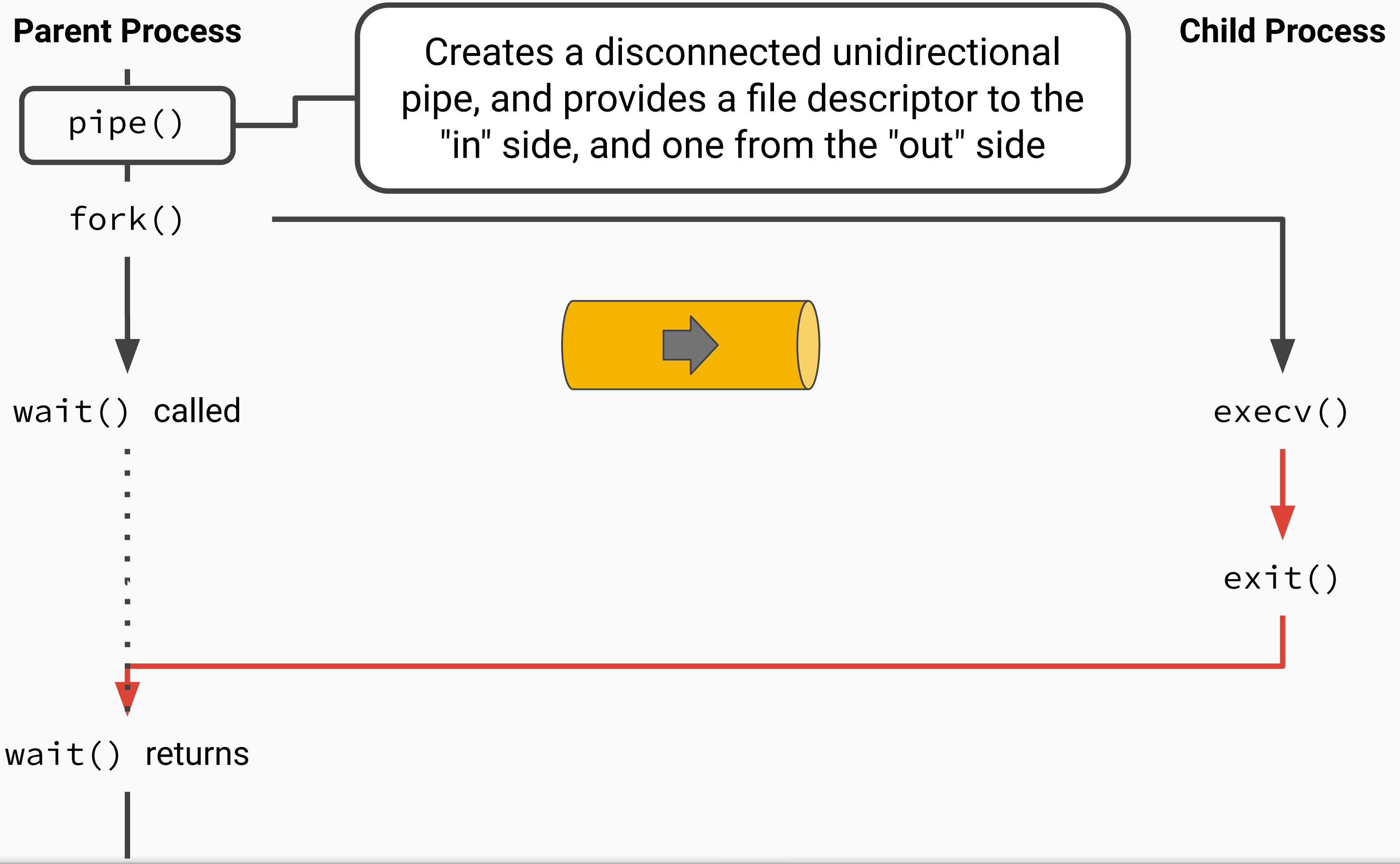
IPC & Processes

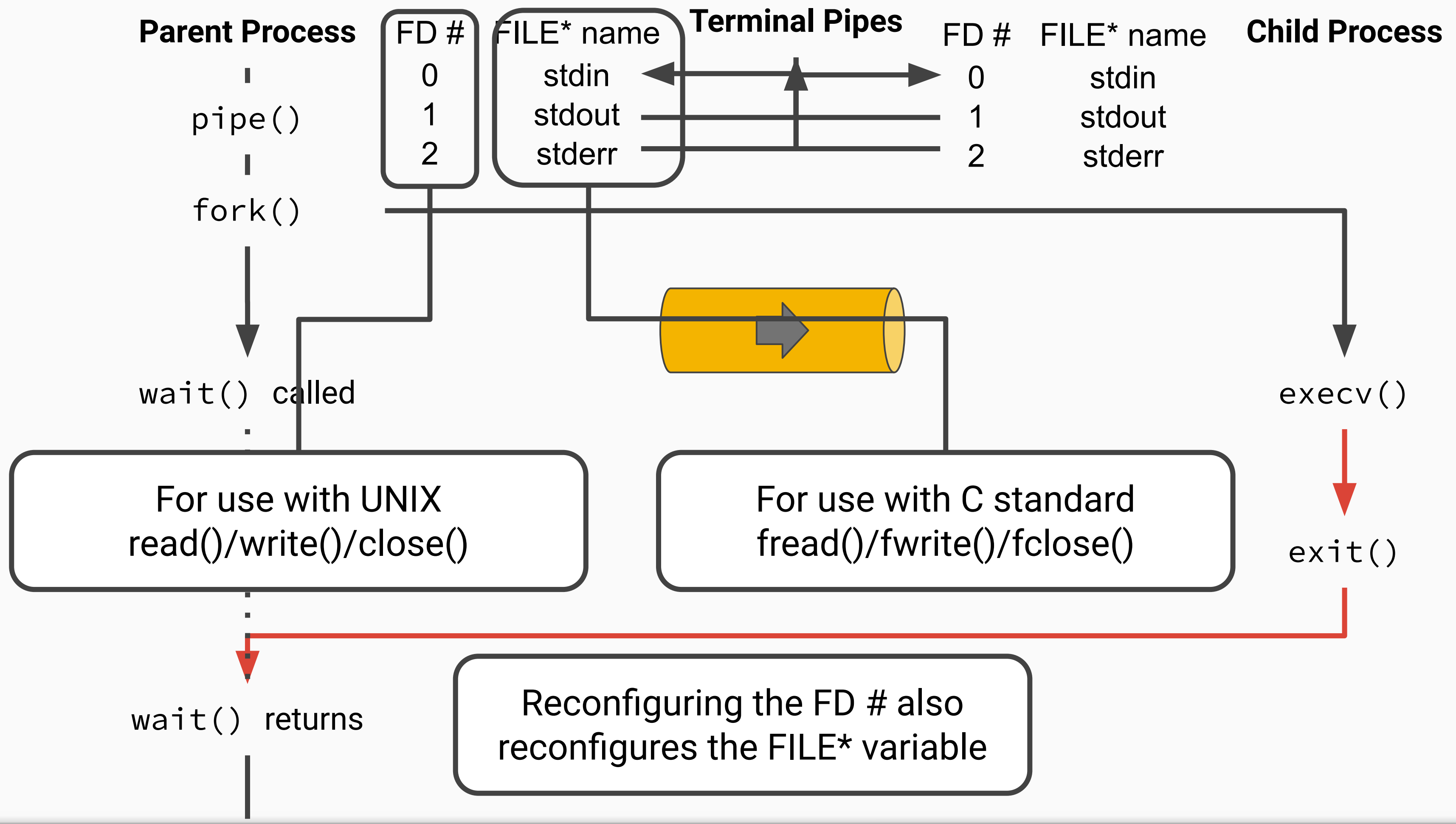
## Waiting for children

We can wait for the child process to complete using wait()




Parent Process
Child Process
fork()
wait() called
execv()
exit()
wait() returns

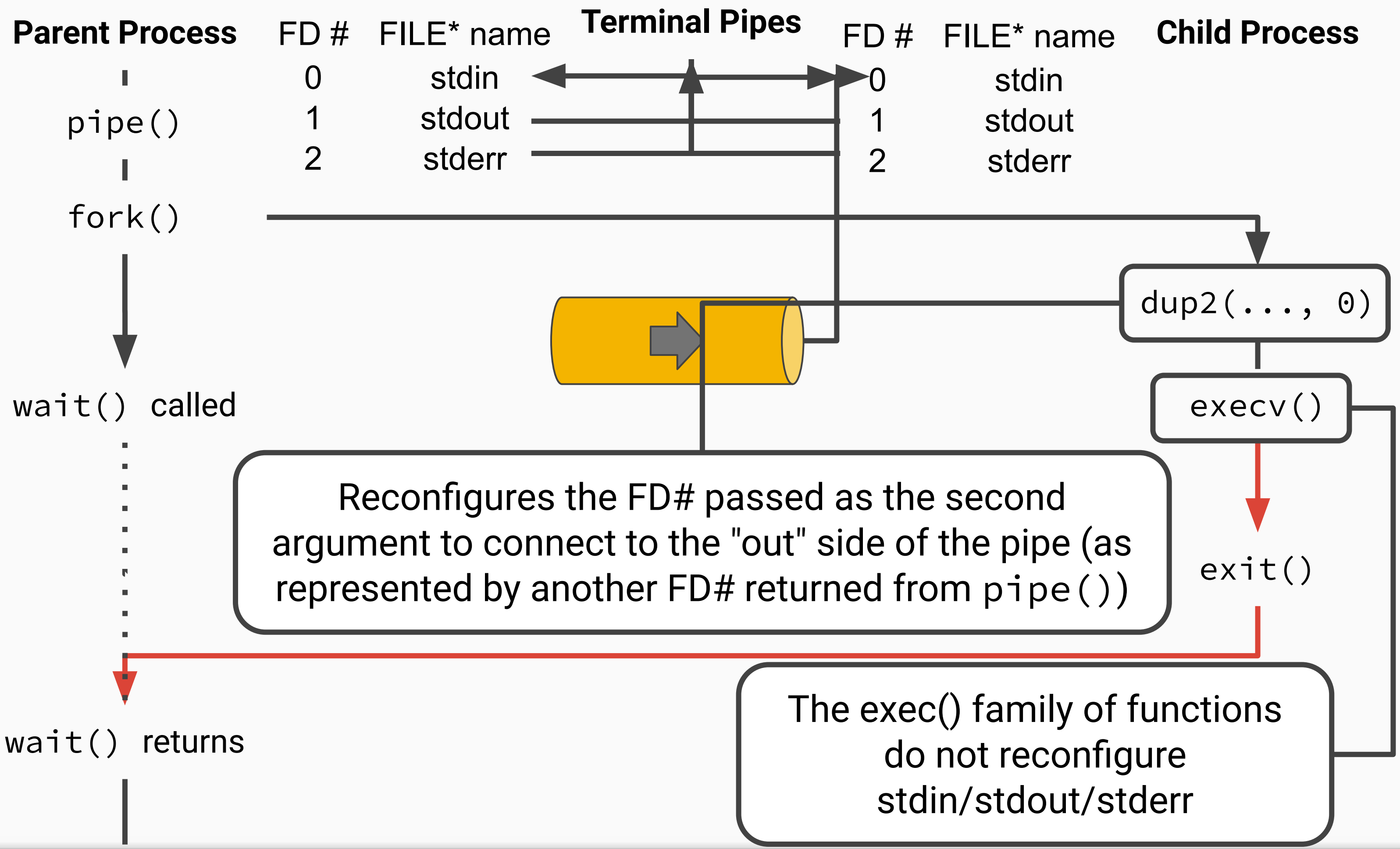

Parent Process
Child Process
pipe()
Creates a disconnected unidirectional pipe, and provides a file descriptor to the "in" side, and one from the "out" side
fork()
wait() called
execv()
exit()
wait() returns



For use with UNIX read()/write()/close()

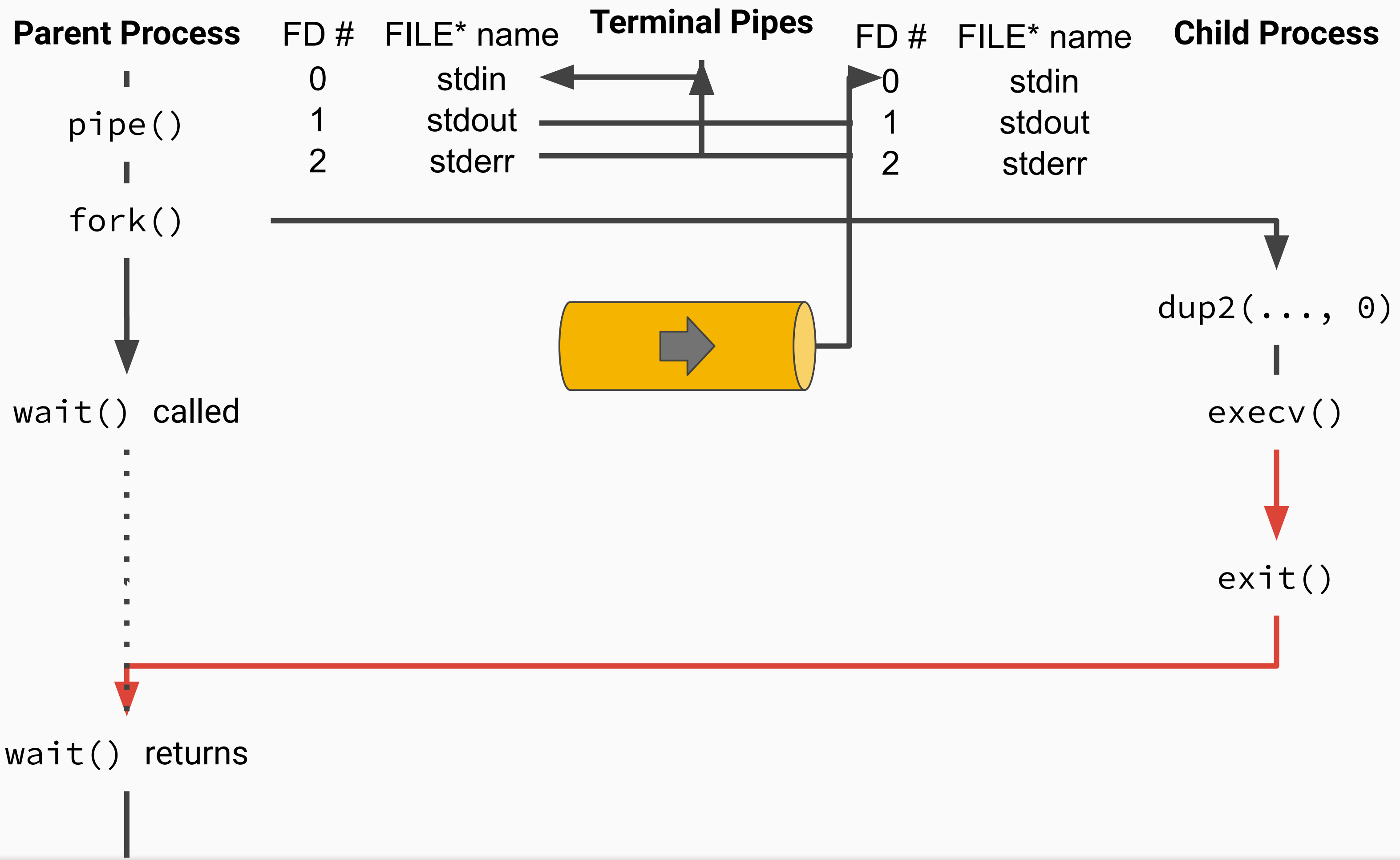
For use with C standard fread()/fwrite()/fclose()

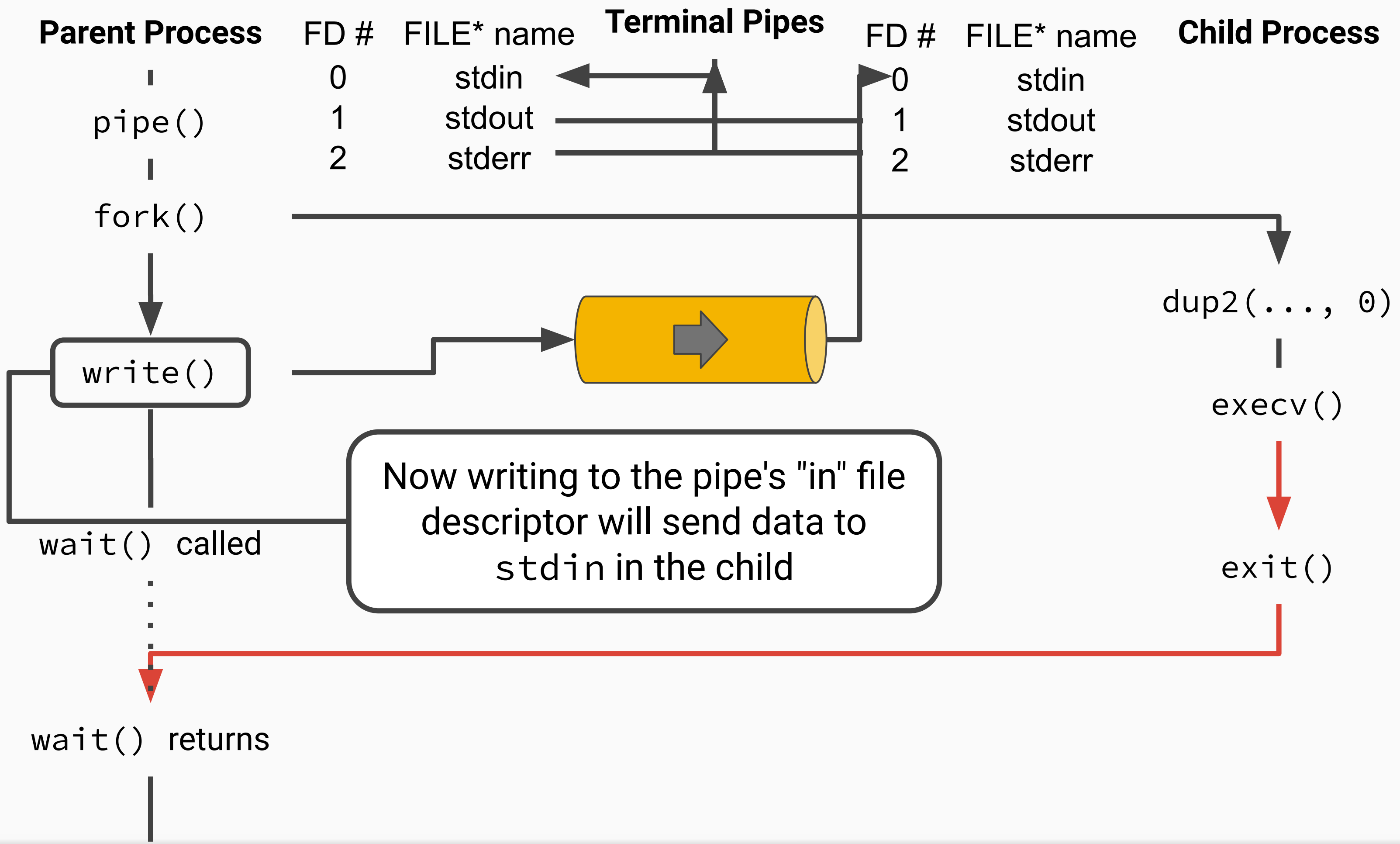
Reconfiguring the FD # also reconfigures the FILE* variable



Reconfigures the FD# passed as the second argument to connect to the "out" side of the pipe (as represented by another FD# returned from `pipe()`)

The exec() family of functions do not reconfigure stdin/stdout/stderr


Parent Process
FD #
FILE* name
Terminal Pipes
FD #
FILE* name
Child Process
pipe()
0
stdin
1
stdout
2
stderr
0
stdin
1
stdout
2
stderr
fork()
dup2(..., 0)
wait() called
execv()
exit()
wait() returns



Now writing to the pipe's "in" file descriptor will send data to `stdin` in the child

IPC & Processes

## Demo resumed

## Final matters

`popen()` can do the `fork()`, `exec()`, and configure one pipe automatically.

- ➔ But, to control both `stdin` *and* `stdout` in the child, do the steps manually
- ➔ Final will expect knowledge of `fork()`, `exec()`, and pipes

One may find `strtok()` useful for argument tokenization (breaking the string into individual pieces to send in `argv`). See the manpage.

## Studying for the Final

## Logistics & Suggestions

Final review session 1:30 PM to 4:00 PM on Saturday, April 29th.

- ➔ Check start and end of assigned chapters, and go from there
  - ◆ Don't miss readings assigned as part of assignments
- ➔ Review your assignment implementations, and try improving them by applying our style and performance feedback

## Questions?

Contact:
Email: hacker@unc.edu
Twitter: @JJBakita
Web: https://cs.unc.edu/~jbakita

Old Well, University of North Carolina at Chapel Hill, Winter 2017