

HTTP & exec

Computer Systems Programming, Spring 2023

Instructor: Travis McGaha

TAs:

Kevin Bernat

Jialin Cai

Mati Davis

Donglun He

Chandravarman Kunjeti

Heyi Liu

Shufan Liu

Eddy Yang



pollev.com/tqm

❖ Any questions from previous lectures?

Logistics

- ❖ HW3 Posted **Due Tonight 4/3 @ 11:59**
 - Should have everything you need
 - Should be on the shorter side theoretically

- ❖ Project Partner Sign Up: **Due Wednesday at midnight**
 - Project spec is up

- ❖ Final Exam Scheduling:
 - 96 hours (4 days)
 - Opens Tuesday May 1st @ Noon
 - Closes Saturday May 6th @ noon

Lecture Outline

- ❖ **Project Demo**
- ❖ HTTP
- ❖ Fork, Pipe, Exec Review

Project: Multi-threaded Search Server

- ❖ Components:
 - Read files and store them into an index
 - Setup a TCP Server Socket
 - Read & Parse HTTP Requests
 - Handle HTTP Requests & send the appropriate response back

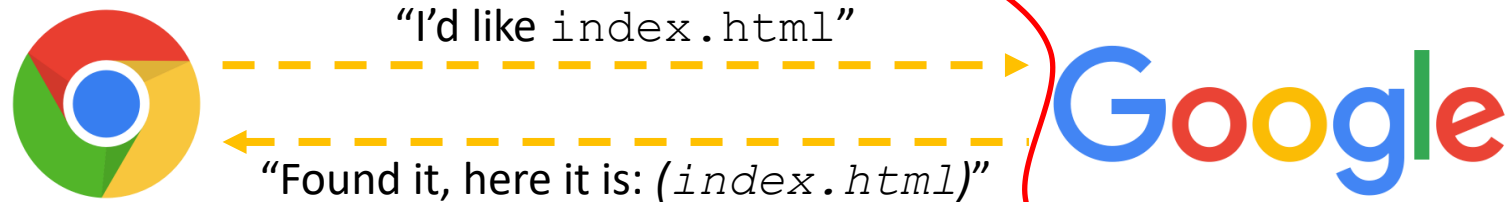
- ❖ Demo: Searching, URL, URI

Lecture Outline

- ❖ Project Demo
- ❖ **HTTP**
- ❖ Fork, Pipe, Exec Review

HTTP Basics

HTTP is part of the application layer
built on top of transport layer



- ❖ A client establishes one or more TCP connections to a server
 - The client sends a request for a web object over a connection and the server replies with the object's contents

- ❖ We have to figure out how to let the client and server communicate their intentions to each other clearly
 - We have to define a *protocol*

Protocols

- ❖ A **protocol** is a set of rules governing the format and exchange of messages in a computing system
 - What messages can a client exchange with a server?
 - What is the syntax of a message?
 - What do the messages mean?
 - What are legal replies to a message?
 - What sequence of messages are legal?
 - How are errors conveyed?

- ❖ A protocol is (roughly) the network equivalent of an API

HTTP

❖ Hypertext Transport Protocol

- A request / response protocol
 - A client (web browser) sends a request to a web server
 - The server processes the request and sends a response
- Typically, a **request** asks a server to retrieve a resource *e.g. a webpage, image, etc*
 - A *resource* is an object or document, named by a Uniform Resource Identifier (URI)
- A **response** indicates whether or not the server succeeded
 - If so, it provides the content of the requested response
- More info: https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol

HTTP Requests

Type of Action to take

❖ General form:

Resource to act on

In this class, 1.1

- `[METHOD]` `[request-uri]` `HTTP/[version]` `\r\n`
`[headerfield1]: [fieldvalue1]` `\r\n`
`[headerfield2]: [fieldvalue2]` `\r\n`
`[...]`
`[headerfieldN]: [fieldvalueN]` `\r\n`

Any# of headers
(designed for
flexibility)

`\r\n`

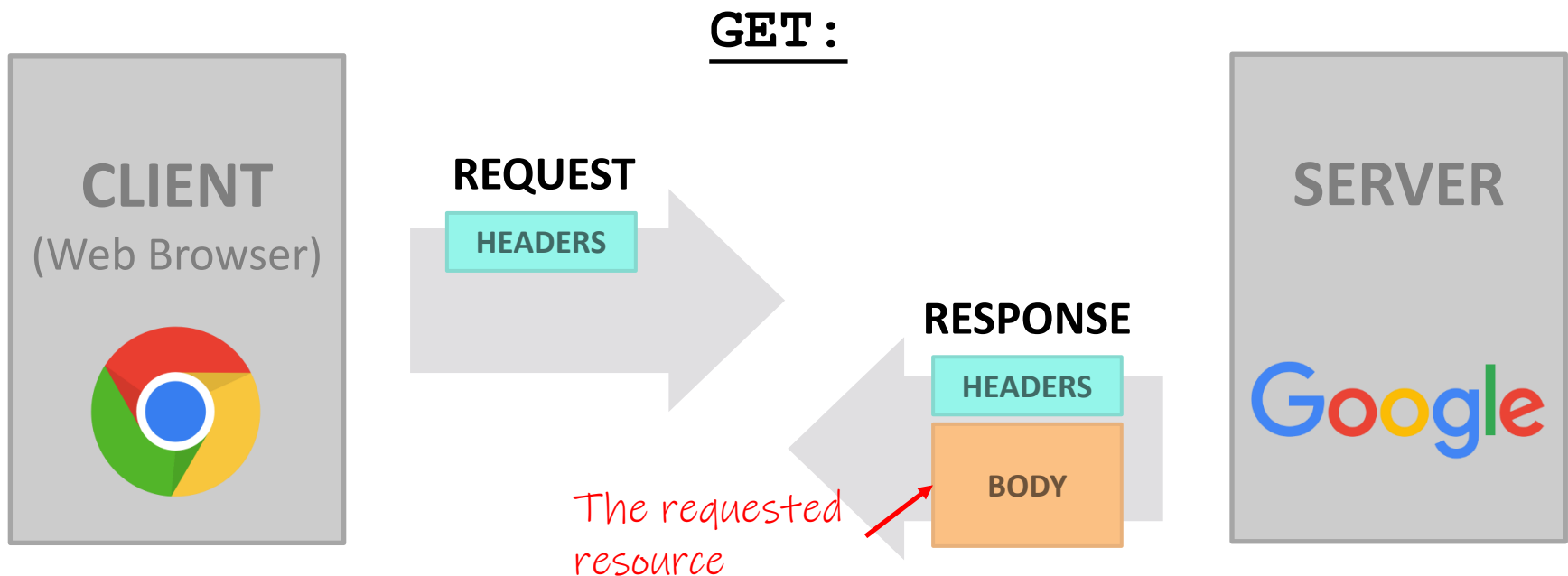
`[request body, if any]`

Blank line
to indicate
the end of
the
headers.

`\r\n` is used to indicate a
"new line" in HTTP

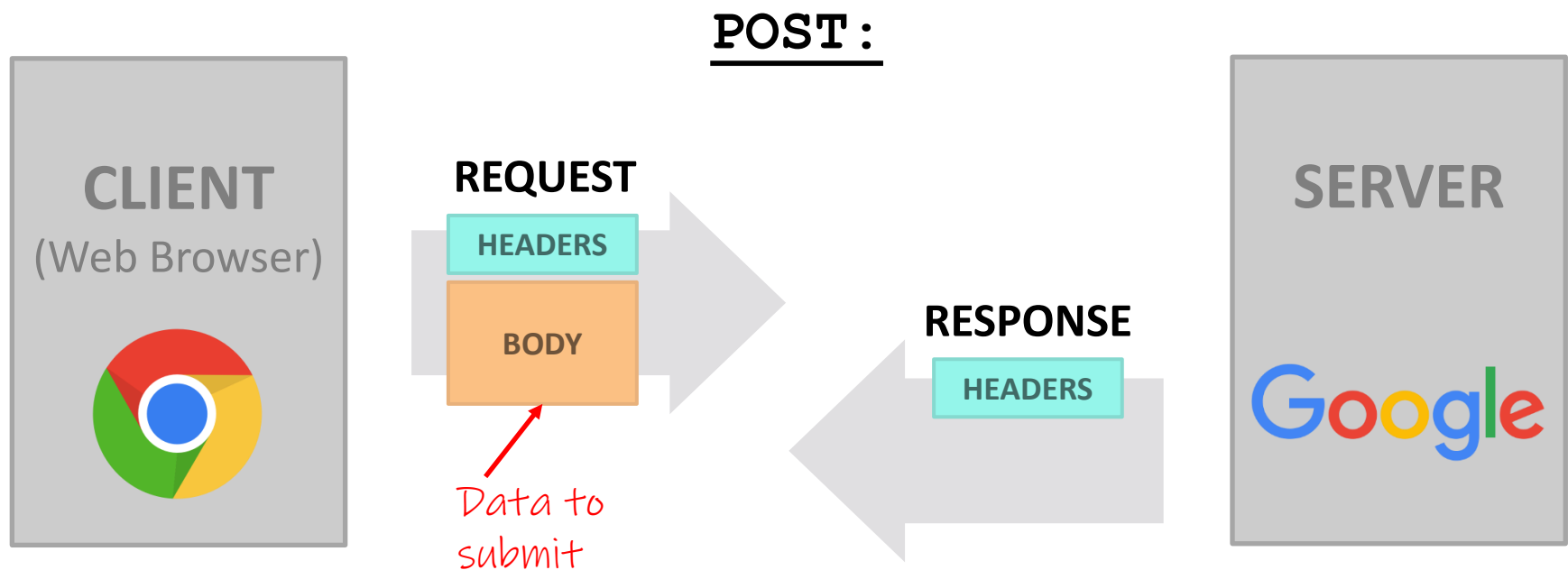
HTTP Methods

- ❖ There are three commonly-used HTTP methods:
 - **GET**: “Please send me the named resource” *Used in the project*



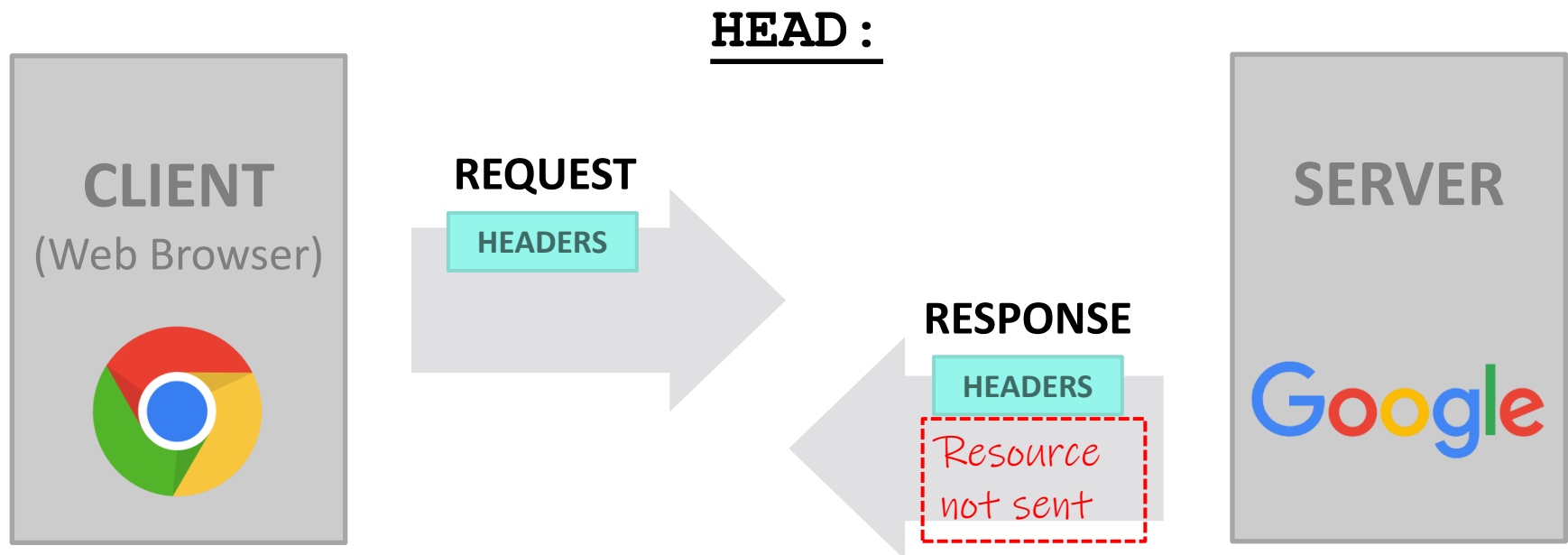
HTTP Methods

- ❖ There are three commonly-used HTTP methods:
 - **GET**: “Please send me the named resource”
 - **POST**: “I’d like to submit data to you” (*e.g.* file upload)



HTTP Methods

- ❖ There are three commonly-used HTTP methods:
 - **GET**: “Please send me the named resource”
 - **POST**: “I’d like to submit data to you” (*e.g.* file upload)
 - **HEAD**: “Send me the headers for the named resource”
 - Doesn’t send resource; often to check if cached copy is still valid



HTTP Methods

- ❖ There are three commonly-used HTTP methods:
 - `GET`: “Please send me the named resource”
 - `POST`: “I’d like to submit data to you” (*e.g.* file upload)
 - `HEAD`: “Send me the headers for the named resource”
 - Doesn’t send resource; often to check if cached copy is still valid
- ❖ Other methods exist, but are much less common:
 - `PUT`, `DELETE`, `TRACE`, `OPTIONS`, `CONNECT`, `PATCH`, . . .
 - For instance: `TRACE` – “show any proxies or caches in between me and the server”

HTTP Uniform Resource Identifier (URI)


❖ Absolute URI

- Composition: `scheme:[//authority]path[?query]`
- Mainly used for communicating via proxy

❖ Most common form of Request-URI

- Composition: `path[?query]`
- Host is specified through headers
- Query is optional
- Path can be empty (just /)

❖ Example Request-URI:

- `/static/test_tree/books/artofwar.txt` `?terms=hello`


HTTP Versions

- ❖ All current browsers and servers “speak” **HTTP/1.1**
 - Version 1.1 of the HTTP protocol
 - <https://www.w3.org/Protocols/rfc2616/rfc2616.html>
 - Standardized in 1997 and meant to fix shortcomings of HTTP/1.0
 - Better performance, richer caching features, better support for multihomed servers, and much more
- ❖ HTTP/2 standardized recently (published in 2015)
 - Allows for higher performance but doesn't change the basic web request/response model
 - Will coexist with HTTP/1.1 for a long time
 - Hard to change/force a switch in the “wild”*

Client Headers

- ❖ The client can provide one or more request “headers”
 - These provide information to the server or modify how the server should process the request
- ❖ You’ll encounter many in practice
 - <https://www.w3.org/Protocols/rfc2616/rfc2616-sec5.html>
 - `Host`: the DNS name of the server *<- server my host multiple domains*
 - `User-Agent`: an identifying string naming the browser *Desktop vs. mobile*
 - `Accept`: the content types the client prefers or can accept
 - `Cookie`: an HTTP cookie previously set by the server

A Real Request

request uri version

`GET / HTTP/1.1`

`Host: breadsouth-turbopromo.codio.io:3333`

`Connection: keep-alive` ← *Keep connection alive after this request*

`Upgrade-Insecure-Requests: 1`

`User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/66.0.3359.181 Safari/537.36`

`Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8` ← *Chrome windows desktop*

`DNT: 1`

`Accept-Encoding: gzip, deflate`

`Accept-Language: en-US,en;q=0.9`

`Cookie: SESS0c8e598bbe17200b27e1d0a18f9a42bb=5c18d7ed6d369d56b69a1c0aa441d78f; SESSd47cbe79be51e625cab059451de75072=d137dbe7bbe1e90149797dcd89c639b1; _sdsat_DMC_or_CCODE=null; _sdsat_utm_source=; _sdsat_utm_medium=; _sdsat_utm_term=; _sdsat_utm_content=; adblock=blocked; s_fid=50771A3AC73B3FFF-3F18AABD559FFB5D; s_cc=true; prev_page=science.%3A%2Fcontent%2F347%2F6219%2F262%2Ftab-pdf; ist_usr_page=1; sat_ppv=79; ajs_anonymous_id=%229225b8cf-6637-49c8-8568-ecb53cfc760c%22; ajs_user_id=null; ajs_group_id=null; __utma=59807807.316184303.1491952757.1496310296.1496310296.1; __utmc=59807807; __utmc=80`

`...`

HTTP Responses

❖ General form:

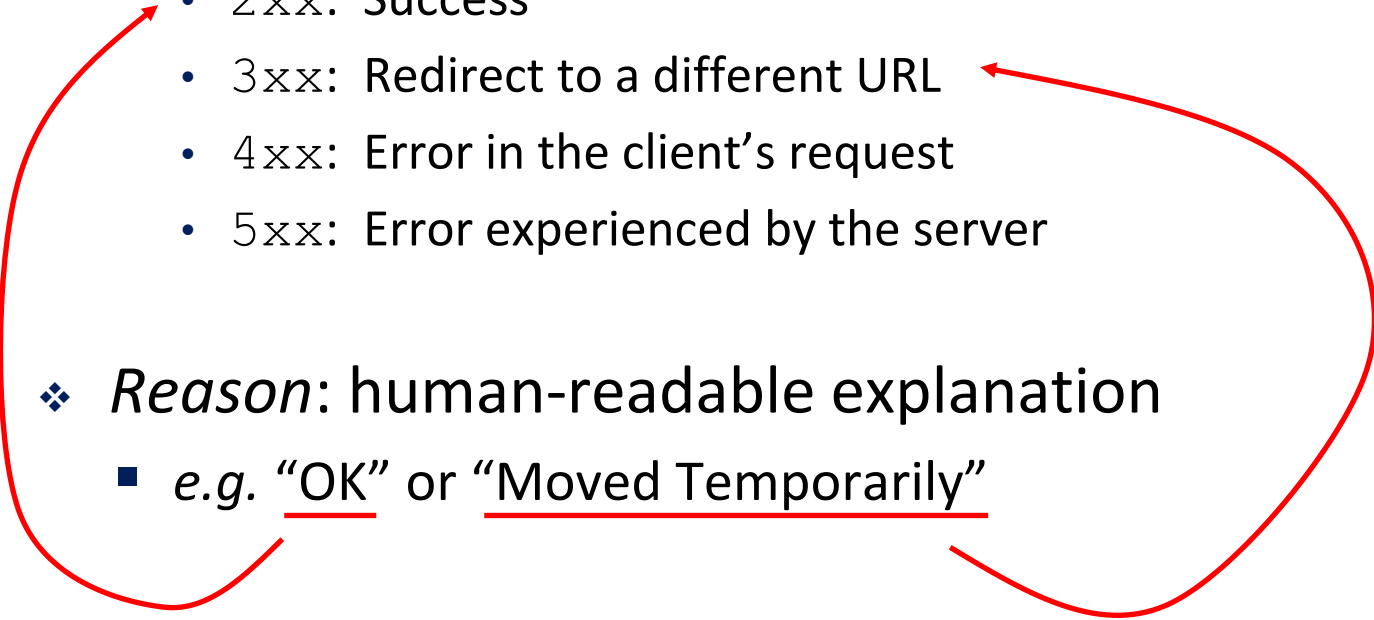
```
■ HTTP/[version] [status code] [reason] \r\n
  [headerfield1]: [fieldvalue1] \r\n
  [headerfield2]: [fieldvalue2] \r\n
  [...]
  [headerfieldN]: [fieldvalueN] \r\n
  \r\n
  [response body, if any]
```

A number

A Human readable string

Typically the requested resource

Status Codes and Reason

- ❖ *Code*: numeric outcome of the request – easy for computers to interpret
 - A 3-digit integer with the 1st digit indicating a response category
 - 1xx: Informational message
 - 2xx: Success
 - 3xx: Redirect to a different URL
 - 4xx: Error in the client's request
 - 5xx: Error experienced by the server
 - ❖ *Reason*: human-readable explanation
 - e.g. “OK” or “Moved Temporarily”
- 

Common Statuses

- ❖ HTTP/1.1 200 OK
 - The request succeeded and the requested object is sent

- ❖ HTTP/1.1 404 Not Found
 - The requested object was not found

- ❖ HTTP/1.1 301 Moved Permanently
 - The object exists, but its name has changed
 - The new URL is given as the “Location:” header value

- ❖ HTTP/1.1 500 Server Error
 - The server had some kind of unexpected error

Server Headers

- ❖ The server can provide zero or more response “headers”
 - These provide information to the client or modify how the client should process the response
- ❖ You’ll encounter many in practice
 - <https://www.w3.org/Protocols/rfc2616/rfc2616-sec6.html>
 - `Server`: a string identifying the server software
 - `Content-Type`: the type of the requested object
 - `Content-Length`: size of requested object
 - `Last-Modified`: a date indicating the last time the request object was modified

*How to interpret
resource
(image, text...)*

When to stop reading

A Real Response

version status reason

```
HTTP/1.1 200 OK
Date: Mon, 21 May 2018 07:58:46 GMT
Server: Apache/2.2.32 (Unix) mod_ssl/2.2.32 OpenSSL/1.0.1e-fips
mod_pubcookie/3.3.4a mod_uwa/3.2.1 Phusion_Passenger/3.0.11
Last-Modified: Mon, 21 May 2018 07:58:05 GMT
ETag: "2299e1ef-52-56cb2a9615625"
Accept-Ranges: bytes
Content-Length: 82
Vary: Accept-Encoding,User-Agent
Connection: close
Content-Type: text/html
Set-Cookie:
bbbbbbbbbbbbbbbb=DBMLFDMJCGAOILMBPIIAAIFLGBAKOJNNMCJIKKBKCDMDEJHMPONHCILPIBL
ADEAKCIABMEEPAOPMMKAOLHOKJMIGMIDKIHNCANAPHMFMBLBABPFENPDANJAPIBOIOOOD;
HttpOnly

<html><body>
<font color="chartreuse" size="18pt">Awesome!!</font>
</body></html>
```

← Length of response body

← Close connection after transaction

← response body is the requested html page

Cool HTTP/1.1 Features

This is extra
(non-testable)
material

- ❖ “Chunked Transfer-Encoding”
 - A server might not know how big a response object is
 - *e.g.* dynamically-generated content in response to a query or other user input
 - How do you send Content-Length?
 - Could wait until you’ve finished generating the response, but that’s not great in terms of *latency* – we want to start sending the response right away
 - Chunked message body: response is a series of chunks

Cool HTTP/1.1 Features

This is extra
(non-testable)
material

❖ Persistent connections

- Establishing a TCP connection is costly
 - Multiple network round trips to set up the TCP connection
 - TCP has a feature called “slow start”; slowly grows the rate at which a TCP connection transmits to avoid overwhelming networks
- A web page consists of multiple objects and a client probably visits several pages on the same server
 - Bad idea: separate TCP connection for each object
 - Better idea: single TCP connection, multiple requests

20 years later...

This is extra
(non-testable)
material

- ❖ World has changed since HTTP/1.1 was adopted
 - Web pages were a few hundred KB with a few dozen objects on each page, now several MB each with hundreds of objects (JS, graphics, ...) & multiple domains per page
 - Much larger ecosystem of devices (phones especially)
 - Many hacks used to make HTTP/1.1 performance tolerable
 - Multiple TCP sockets from browser to server
 - Caching tricks; JS/CSS ordering and loading tricks; cookie hacks
 - Compression/image optimizations; splitting/sharding requests
 - etc., etc. ...

HTTP/2

This is extra
(non-testable)
material

- ❖ Based on Google SPDY; standardized in 2015
 - Binary protocol - easier parsing by machines (harder for humans); sizes in headers, not discovered as requests are processed; ...
 - But same core request/response model (GET, POST, OK, ...)
 - Multiple data streams multiplexed on single TCP connections
 - Header compression, server push, object priorities, more...
- ❖ All existing implementations incorporate TLS encryption (https)
- ❖ Supported by all major browsers and servers since ~2015
- ❖ Used now by most major web sites
 - Coexists with HTTP/1.1
 - HTTP/2 used automatically when browser and server both support it

 **Poll Everywhere**pollev.com/tqm

❖ Are the following statements True or False?

Q1 **Q2**

A. False False

B. False True

C. True False

D. True True

E. We're lost...

Q1: A protocol only defines the “syntax” that clients and servers can communicate with.

Q2: Clients and servers use the same header fields.

 **Poll Everywhere**pollev.com/tqm

❖ Are the following statements True or False?

Q1 Q2

A. False False

B. False True

C. True False

D. True True

E. We're lost...

Q1: A protocol only defines the “syntax” that clients and servers can communicate with. *Also the semantics/meaning*

Q2: Clients and servers use the same header fields.

 **Poll Everywhere**pollev.com/tqm

- ❖ Which HTTP status code family do you think the following Reasons belong to?

Q1 Q2

A. 4xx 2xx

B. 4xx 3xx

C. 5xx 2xx

D. 5xx 3xx

E. We're lost...

Q1: Gateway Time-out

Q2: No Content

Poll Everywhere

pollev.com/tqm

- ❖ Which HTTP status code family do you think the following Reasons belong to?

1xx: info

2xx: success

3xx: redirect

4xx: client fail

5xx: server fail

Q1

Q2

A. 4xx 2xx

B. 4xx 3xx

C. 5xx 2xx

D. 5xx 3xx

E. We're lost...

Q1: Gateway Time-out

Server acting as gateway timed out

Q2: No Content

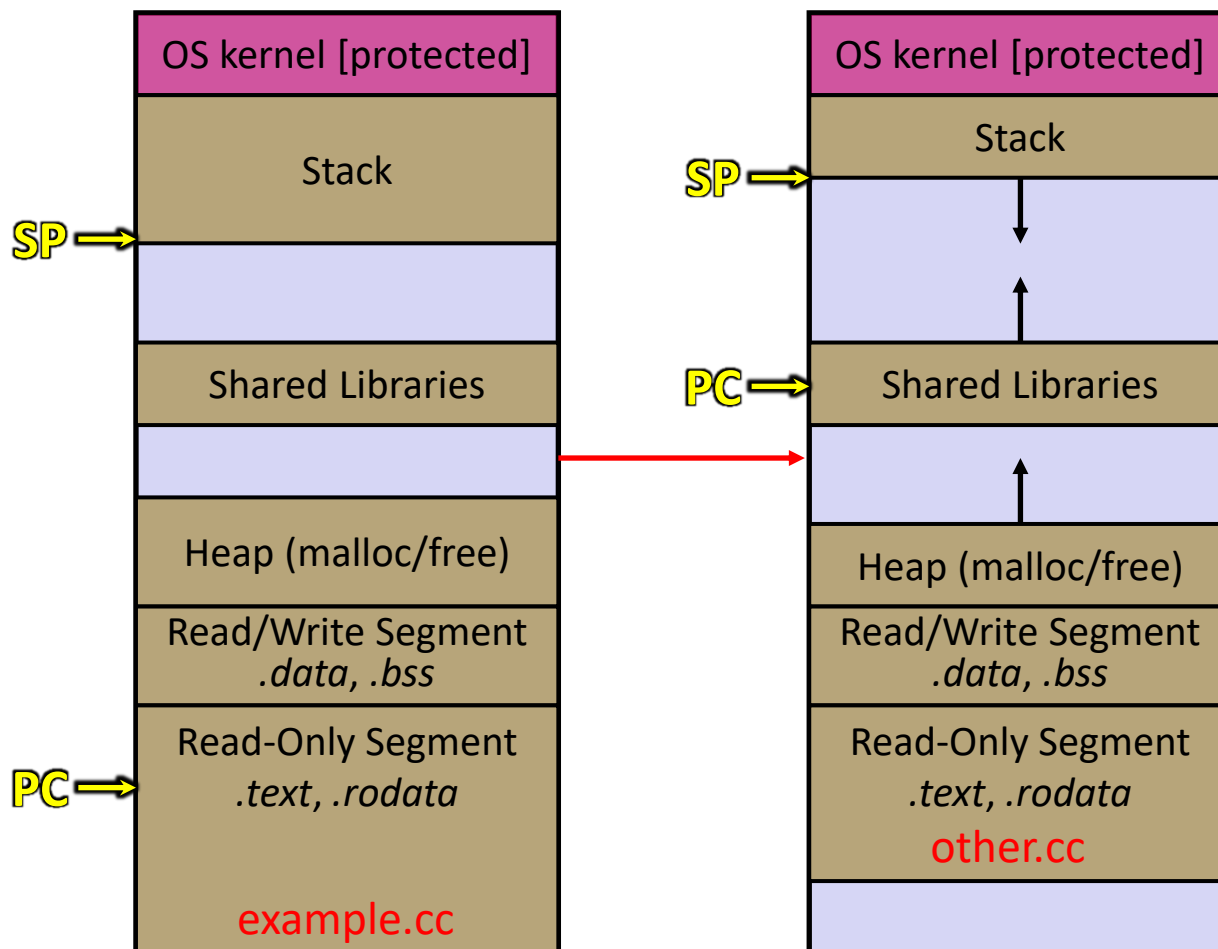
Ok! Resource retrieved, but it is empty

Lecture Outline

- ❖ Project Demo
- ❖ HTTP
- ❖ **Fork, Pipe, Exec Review**

Exec Visualization

- ❖ Exec takes a process and discards or “resets” most of it



NOTE that the following DO change

- The stack
- The heap
- Globals
- Loaded code
- Registers

NOTE that the following do NOT change

- Process ID
- Open files
- The kernel

Exec Demo

- ❖ See `exec_example.cc`
 - Brief code demo to see how exec works
 - What happens when we call exec?
 - What happens if we open some files before exec?
 - What happens if we replace stdout with a file?

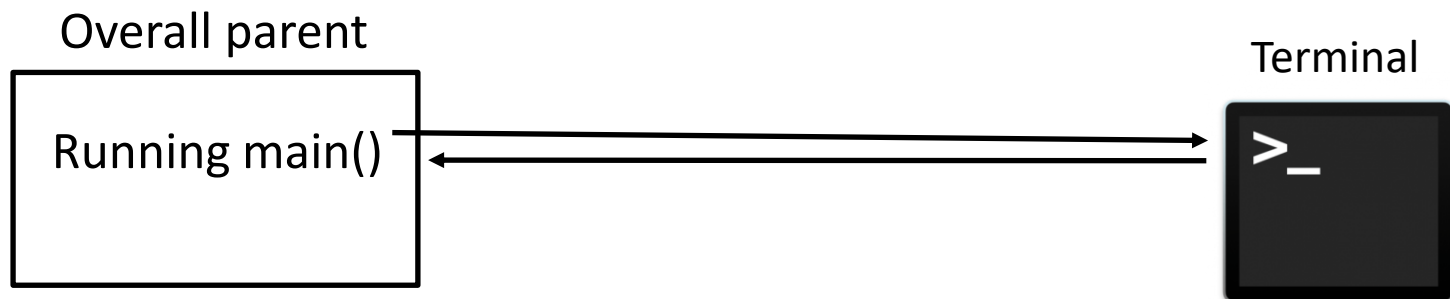
- ❖ NOTE: When a process exits, then it will close all of its open files by default

Exec Demo

- ❖ See `autograder.cc`
 - How could we take advantage of `exec` to do something useful?
 - Combine usage of `fork` and `exec` so our program can do multiple things

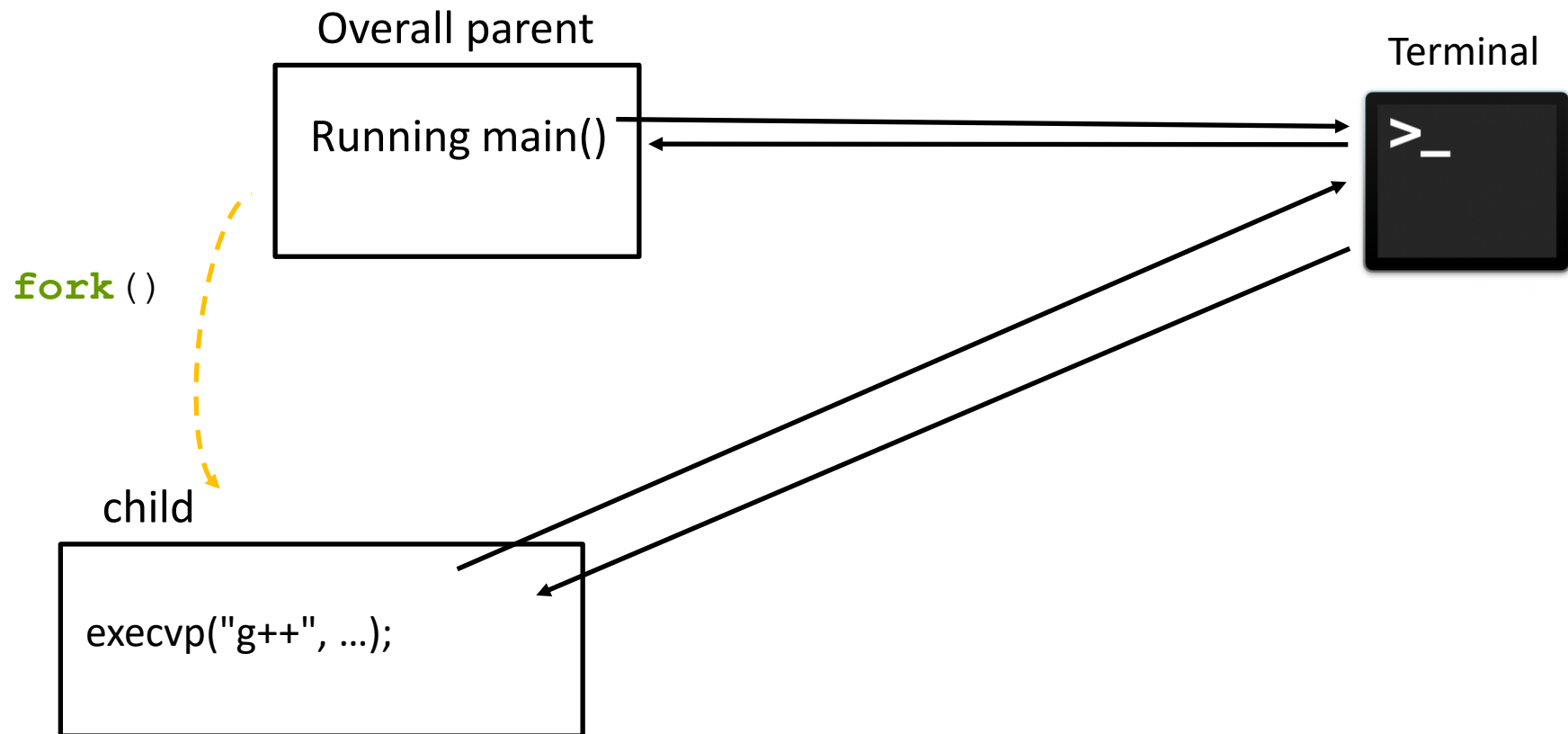
autograder.cc Trace

- ❖ First:
we compile the program with the g++ command



autograder.cc Trace

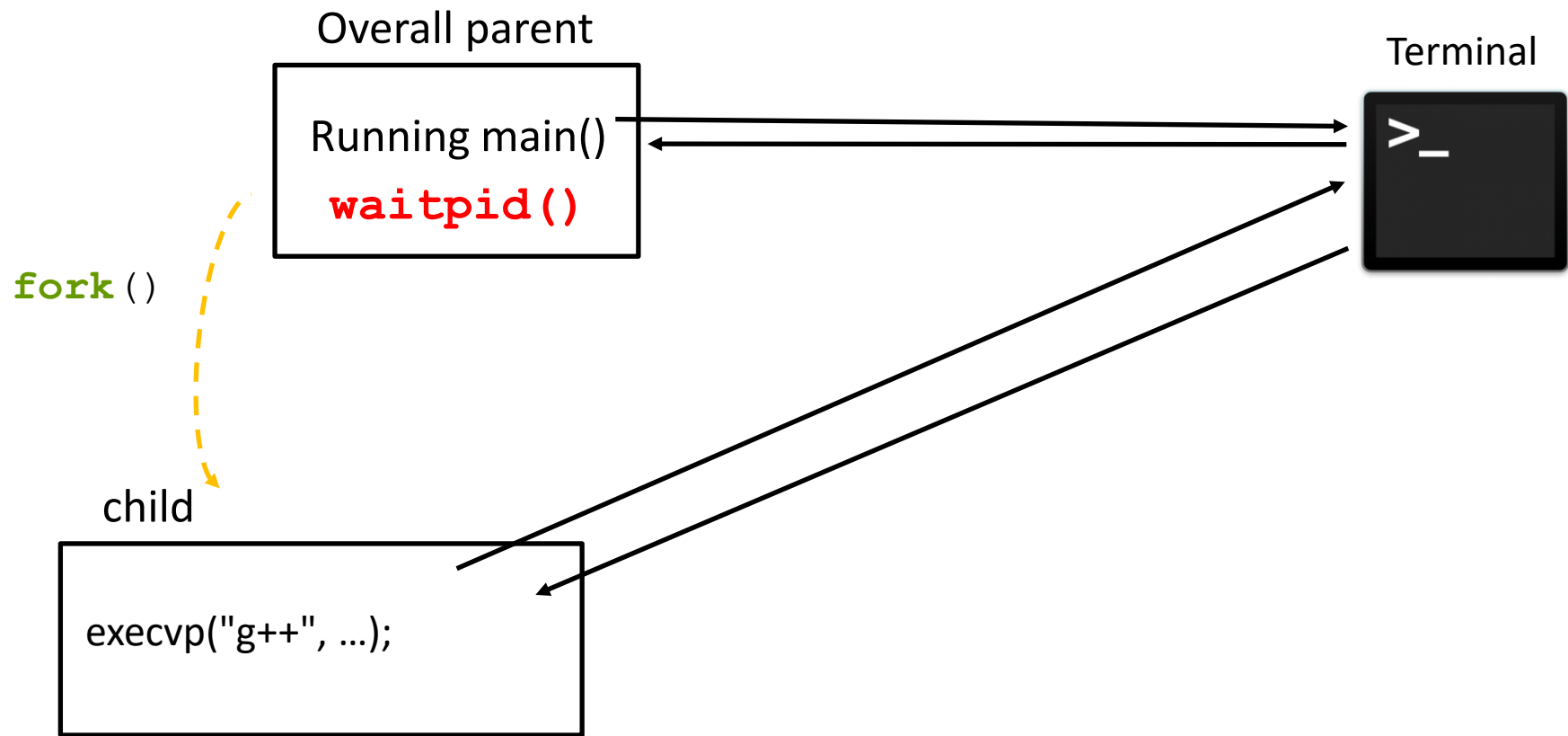
- ❖ First:
 - we compile the program with the g++ command



autograder.cc Trace

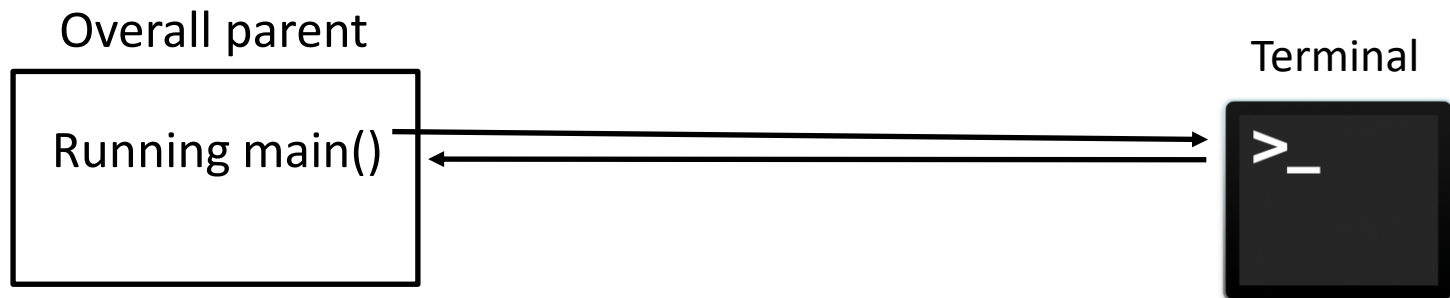
❖ First:

we compile the program with the g++ command



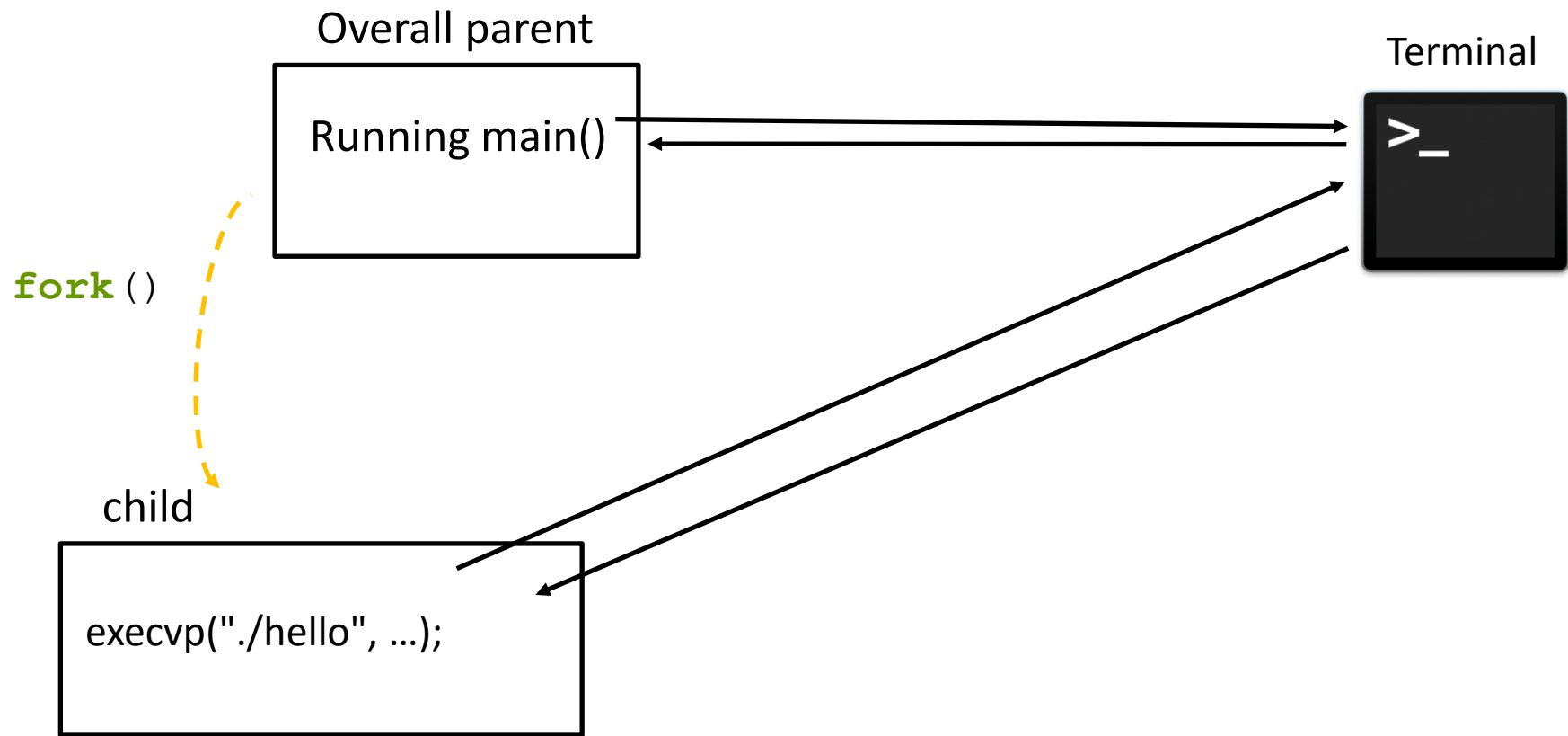
autograder.cc Trace

- ❖ Compilation done!
Run the compiled program



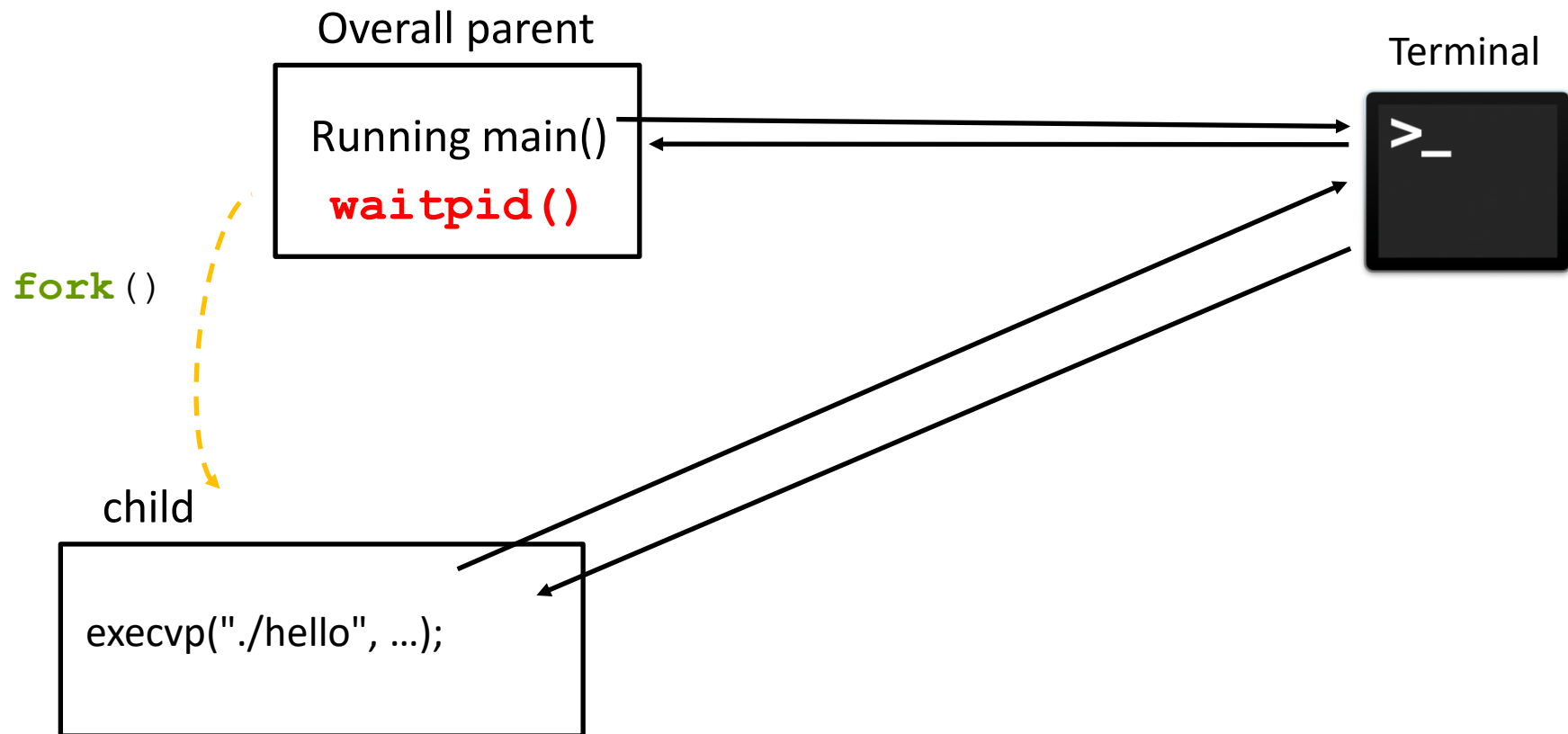
autograder.cc Trace

- ❖ Compilation done!
Run the compiled program



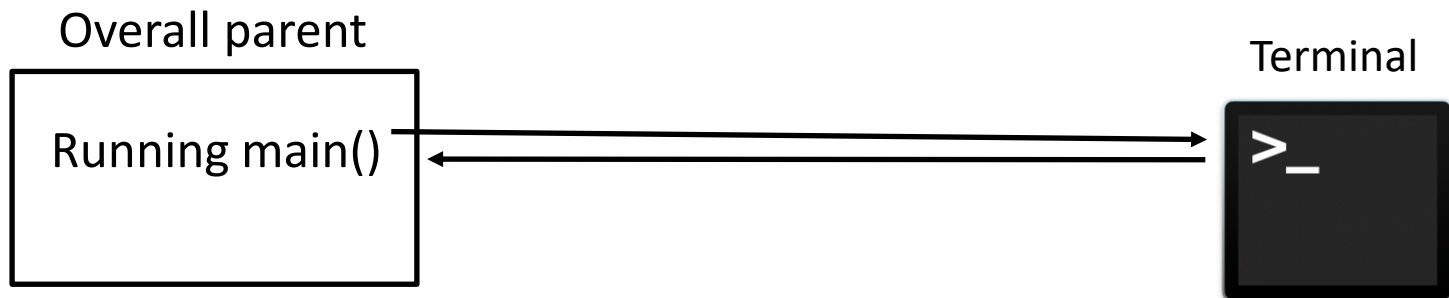
autograder.cc Trace

- ❖ Compilation done!
Run the compiled program



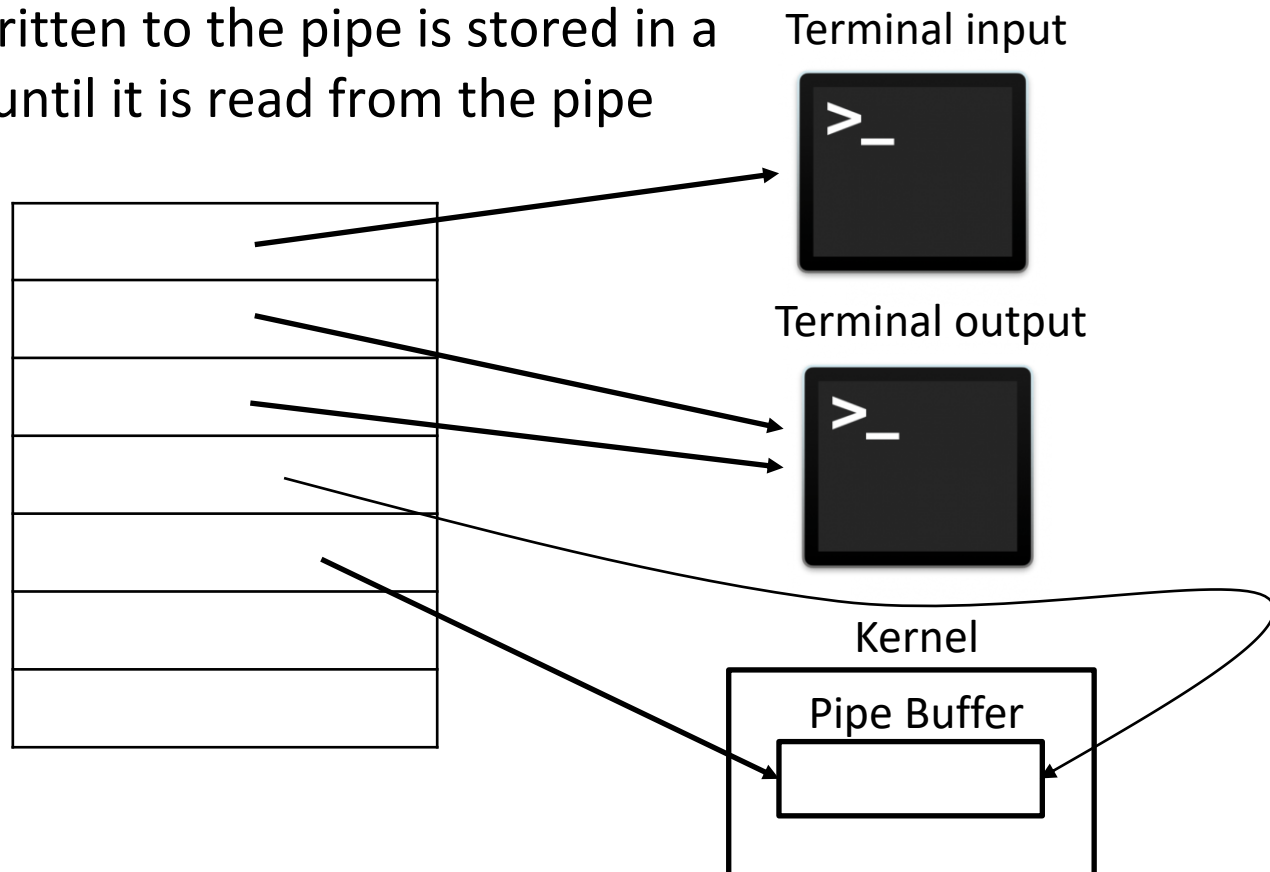
autograder.cc Trace

❖ Done?



Pipe Visualization

- ❖ A pipe can be thought of as a "file" that has distinct file descriptors for reading and writing. This "file" only exists as long as the pipe exists and is maintained by the OS.
 - Data written to the pipe is stored in a buffer until it is read from the pipe

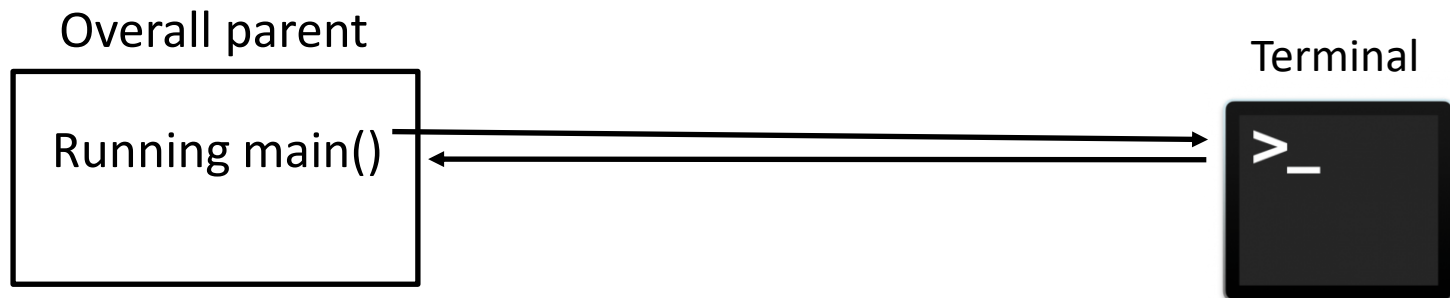


Exec & Pipe Demo

- ❖ See `io_autograder.cc`
 - How could we take advantage of exec and pipe to do something useful?
 - Combine usage of fork and exec so our program can do multiple things

io_autograder.cc Trace

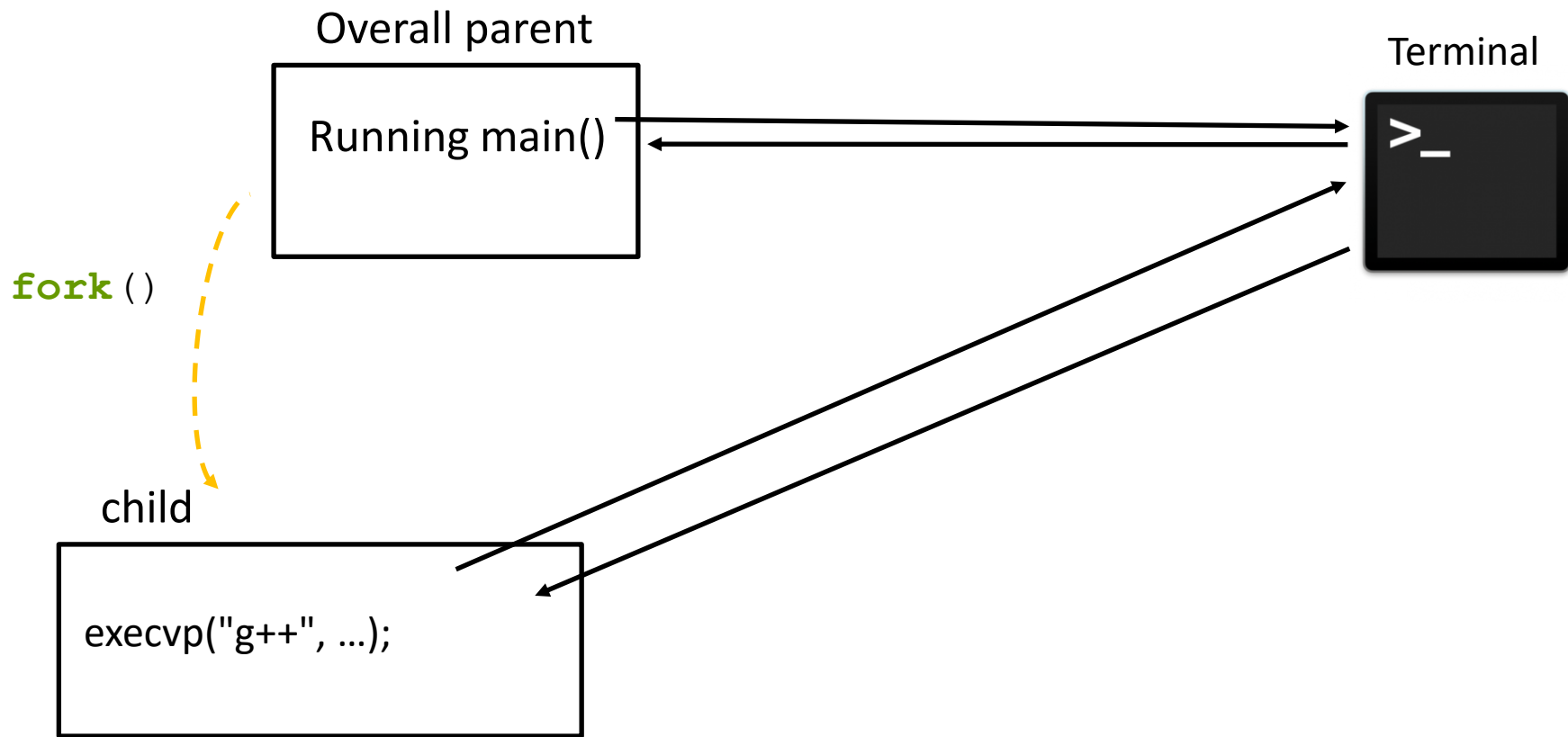
- ❖ First:
we compile the program with the g++ command



io_autograder.cc Trace

❖ First:

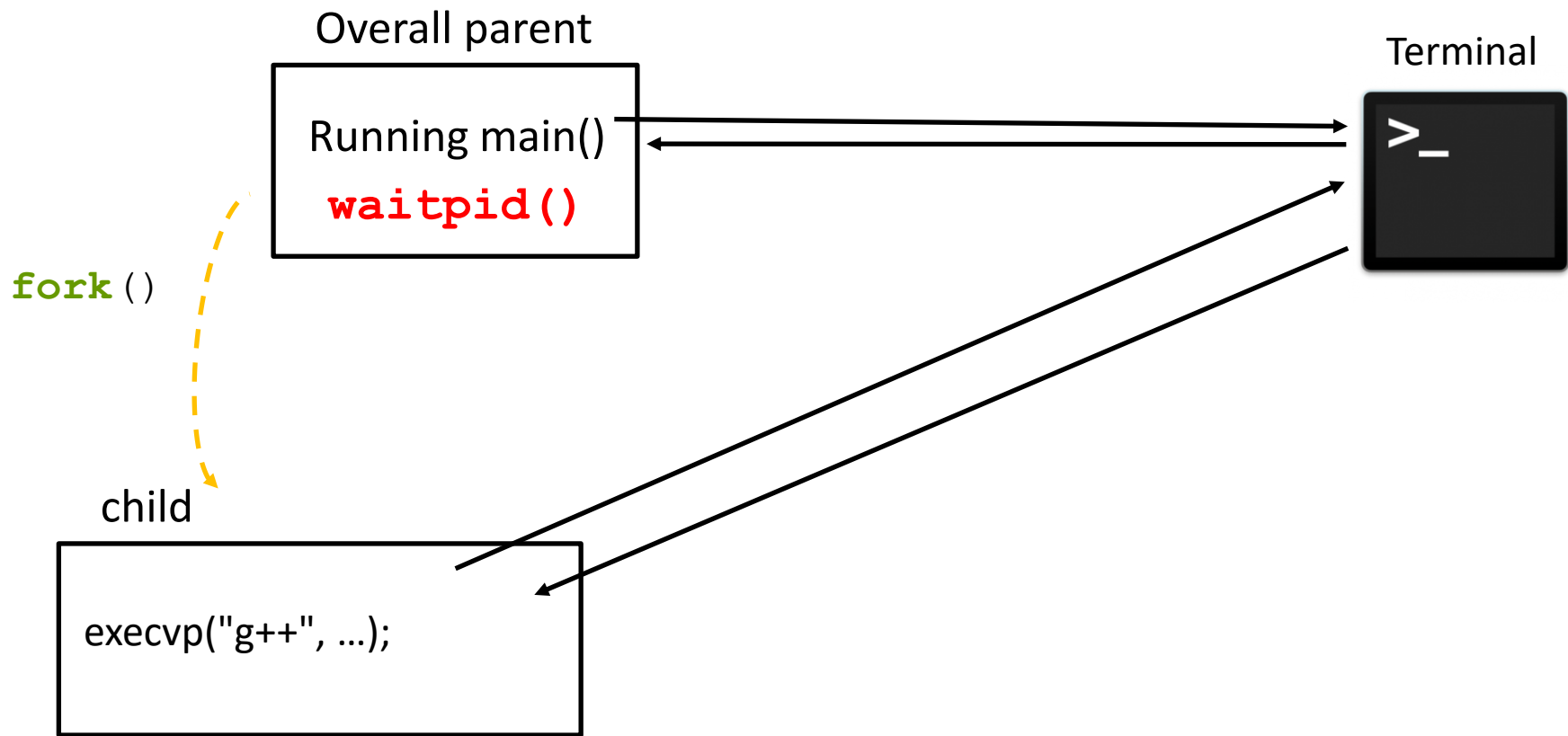
we compile the program with the g++ command



io_autograder.cc Trace

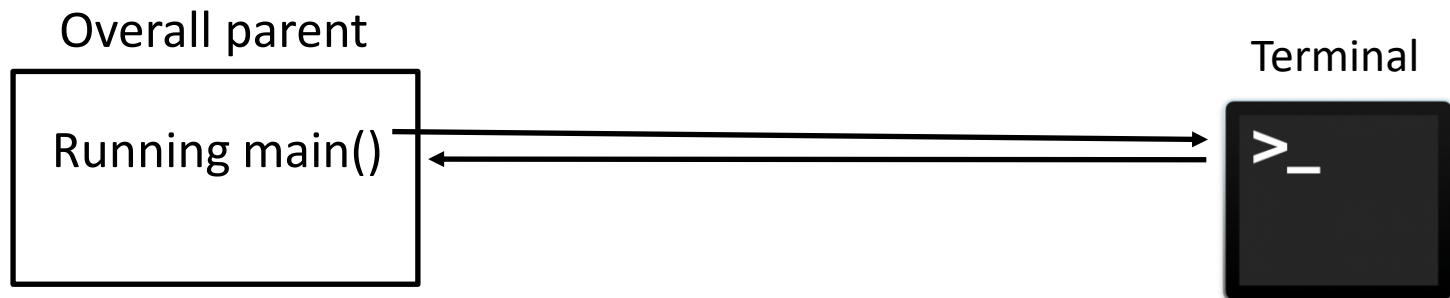
❖ First:

we compile the program with the g++ command



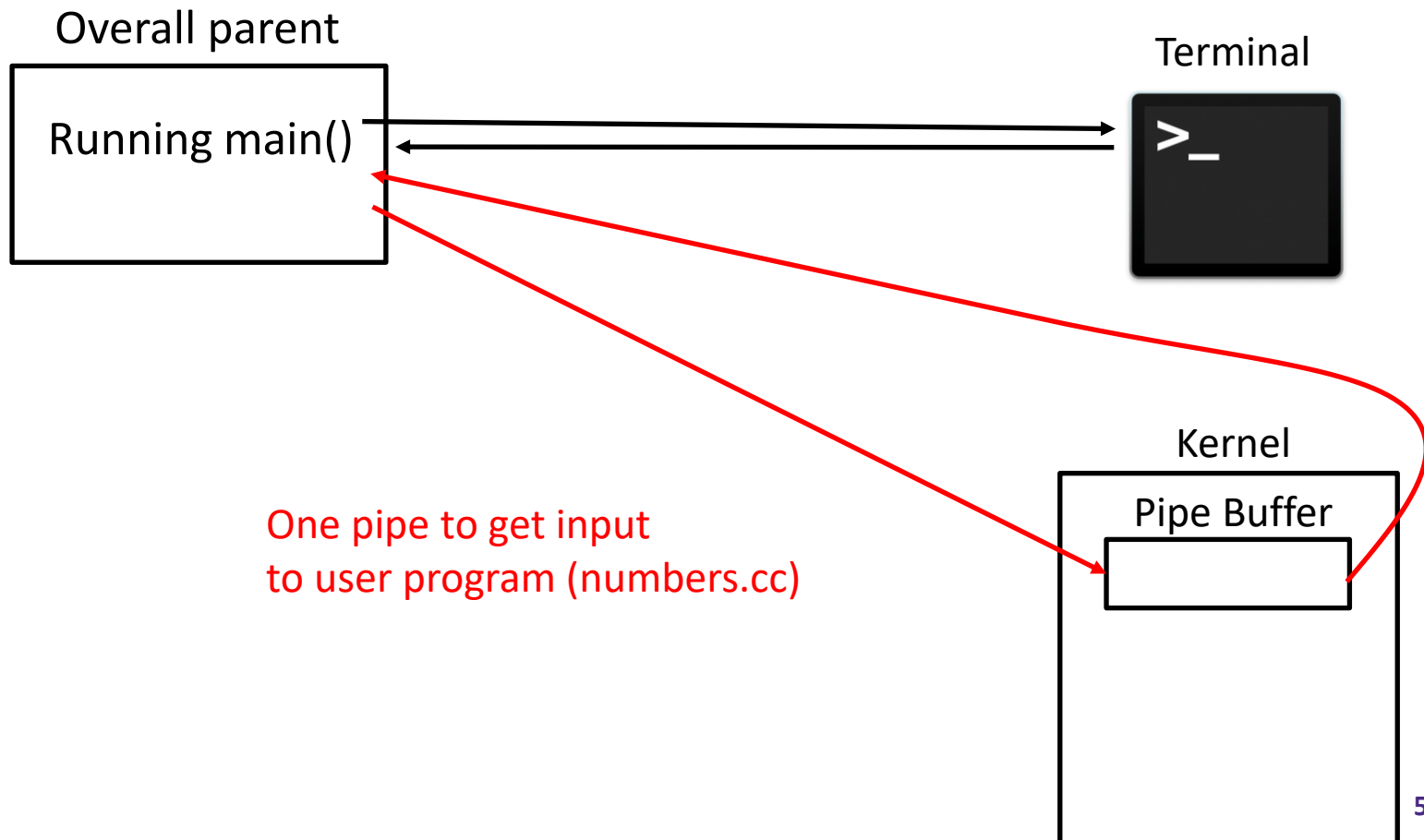
io_autograder.cc Trace

- ❖ Compilation done! Run the compiled program...
BUT send autograder input and capture output



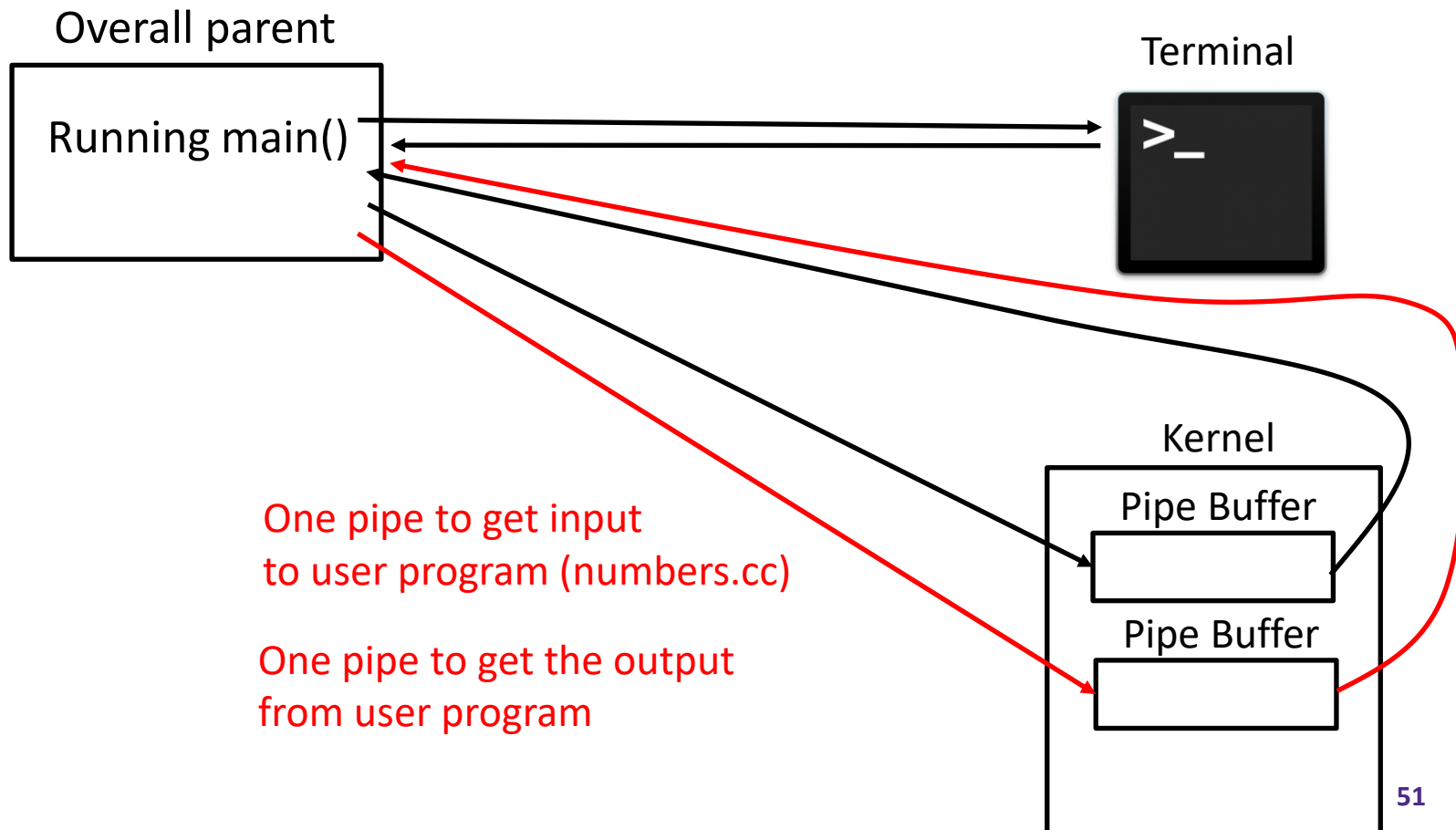
io_autograder.cc Trace

- ❖ Compilation done! Run the compiled program... BUT send autograder input and capture output



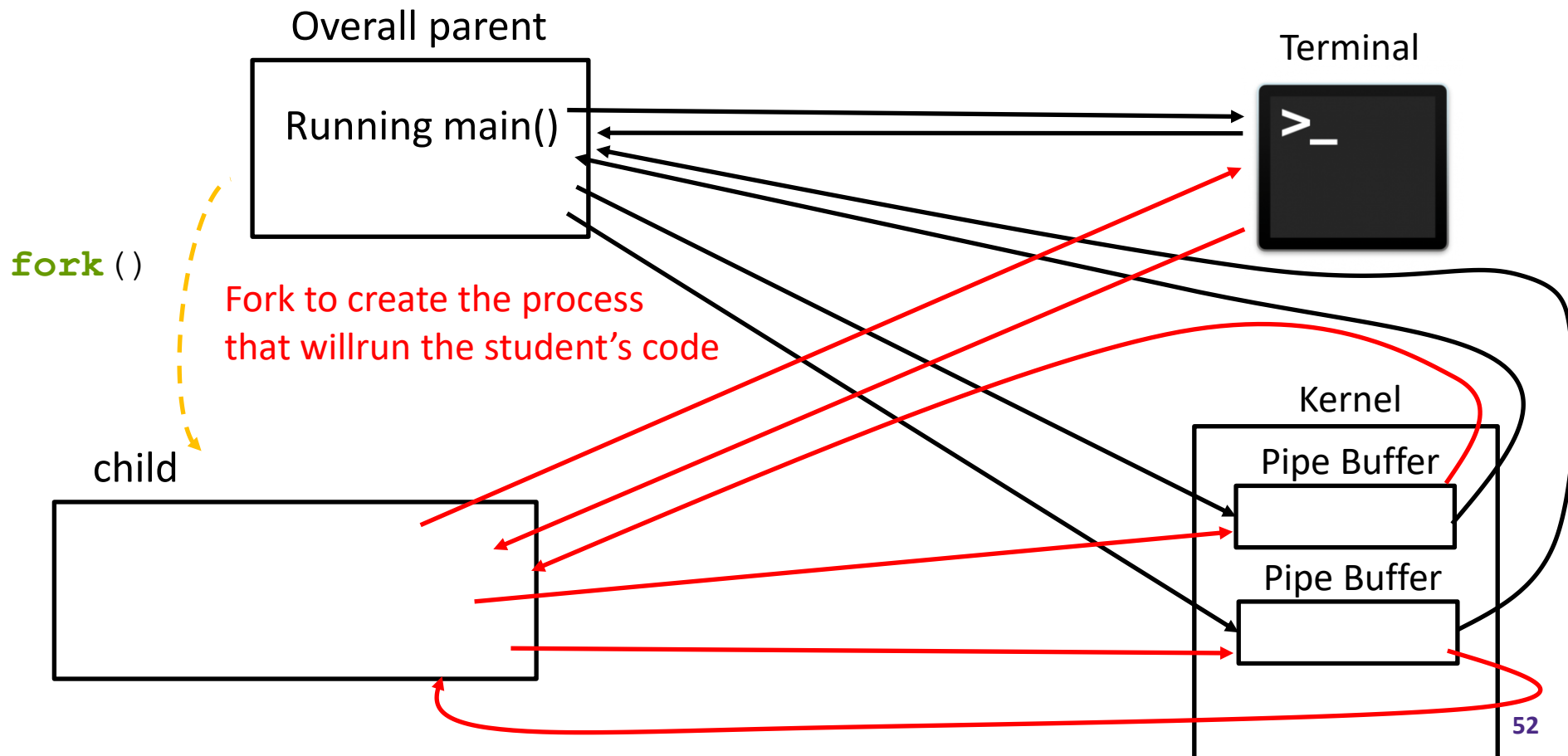
io_autograder.cc Trace

- ❖ Compilation done! Run the compiled program... BUT send autograder input and capture output



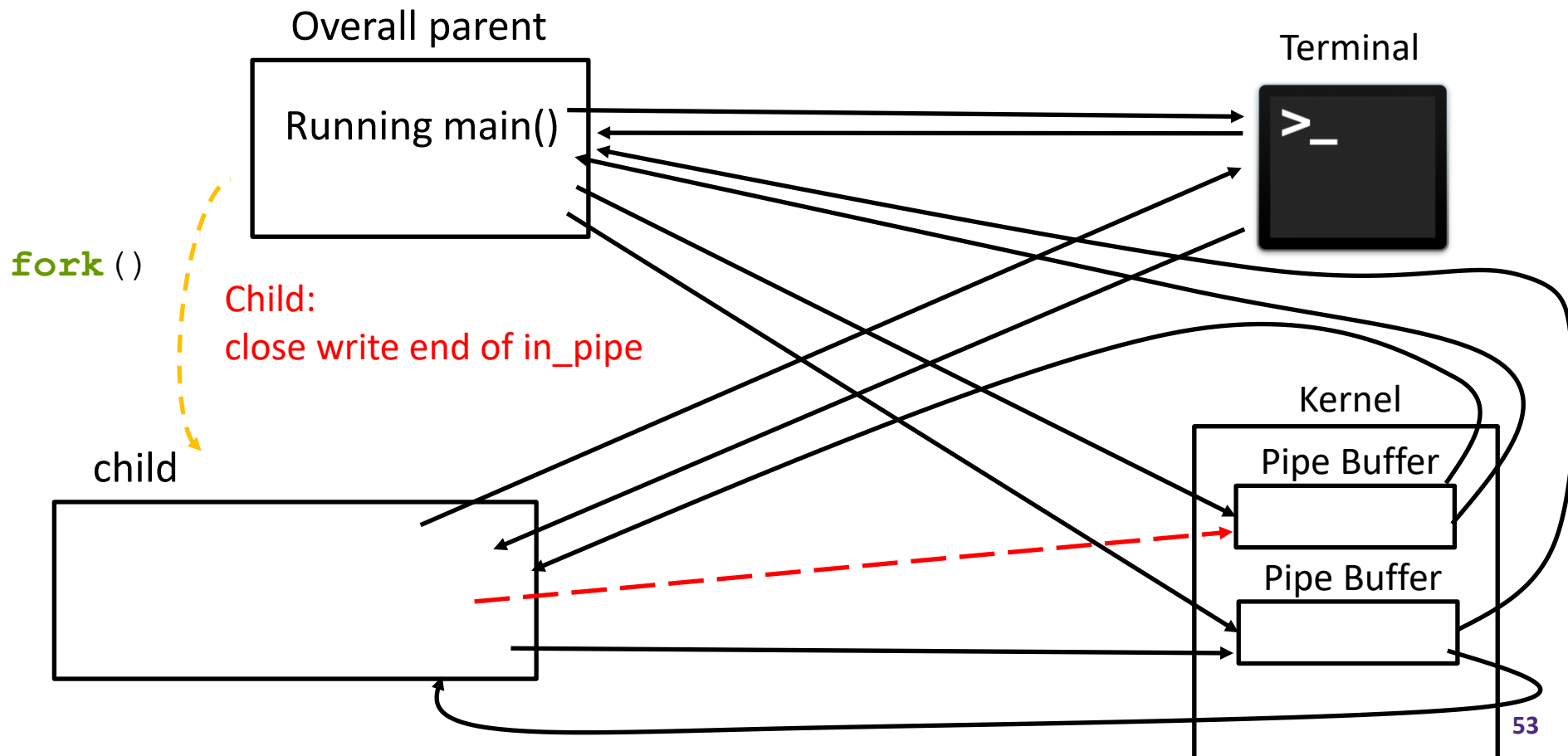
io_autograder.cc Trace

- ❖ Compilation done! Run the compiled program... BUT send autograder input and capture output



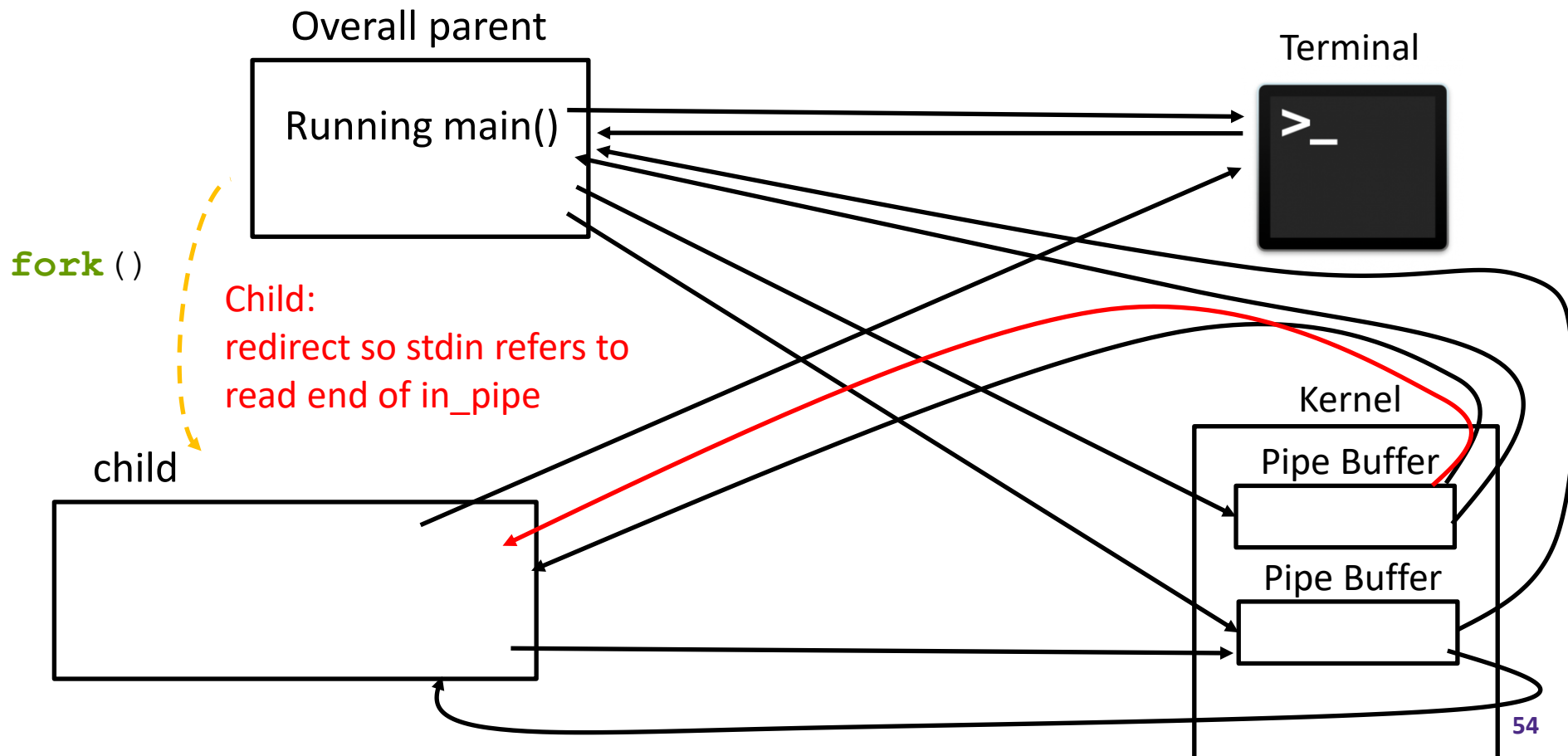
io_autograder.cc Trace

- ❖ Compilation done! Run the compiled program... BUT send autograder input and capture output



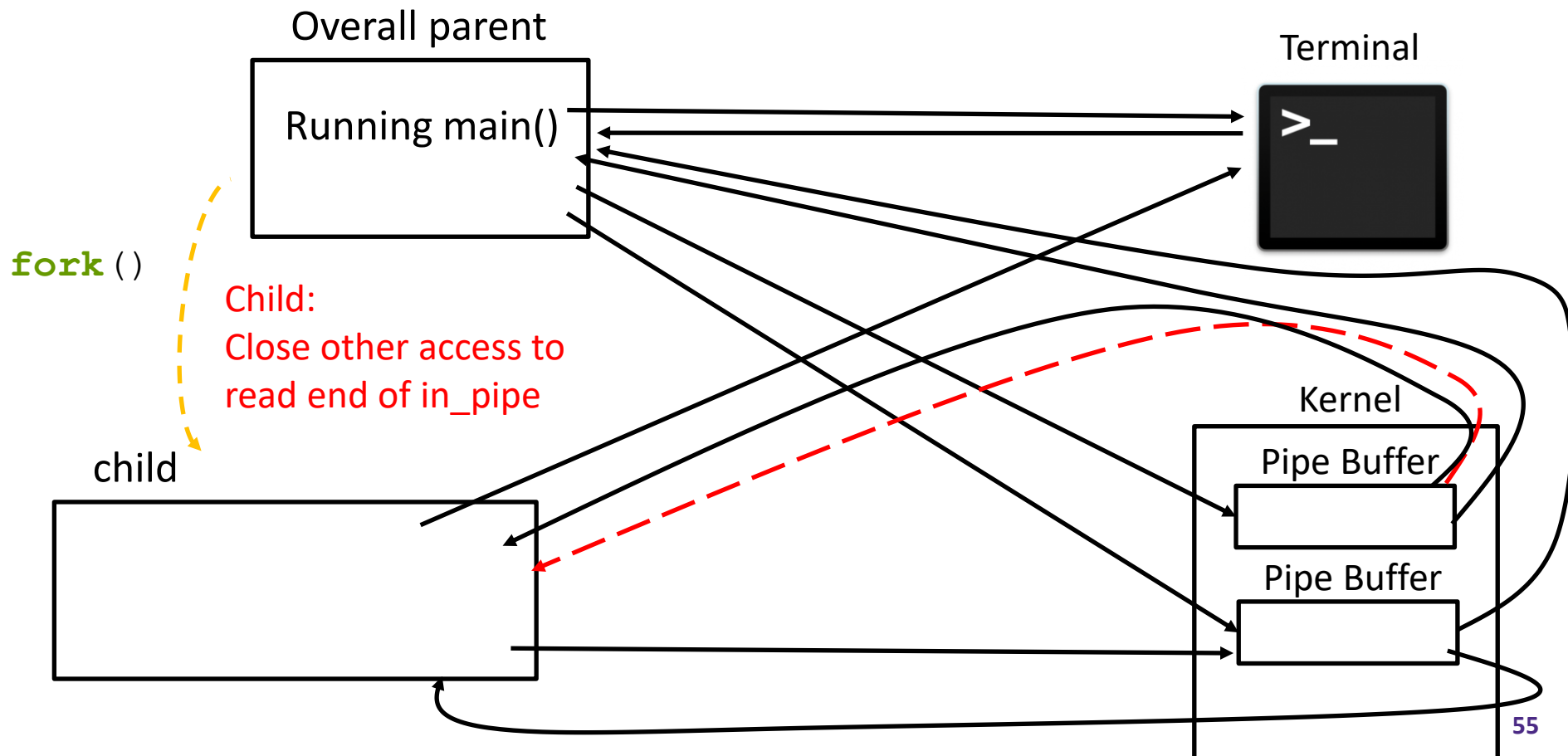
io_autograder.cc Trace

- ❖ Compilation done! Run the compiled program... BUT send autograder input and capture output



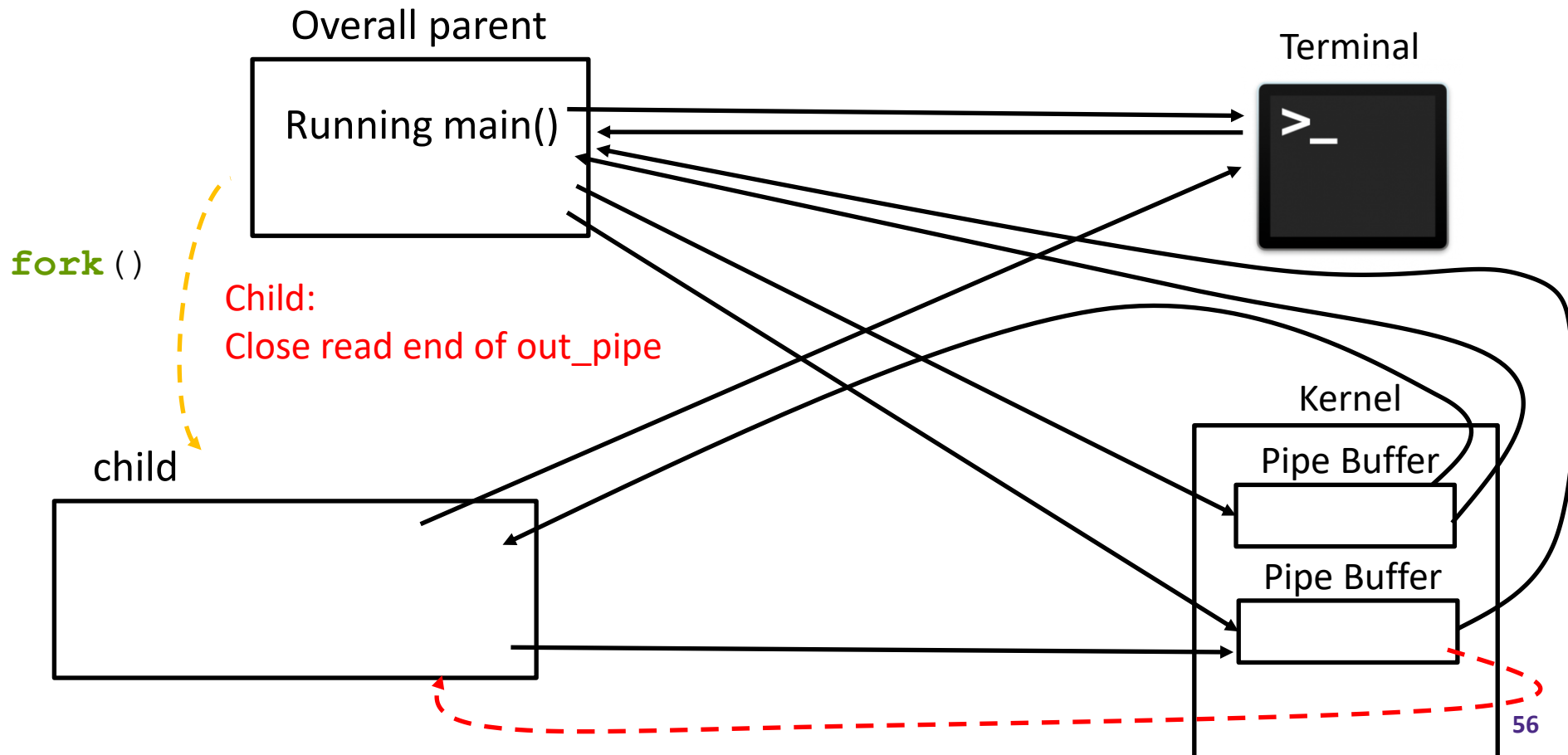
io_autograder.cc Trace

- ❖ Compilation done! Run the compiled program... BUT send autograder input and capture output



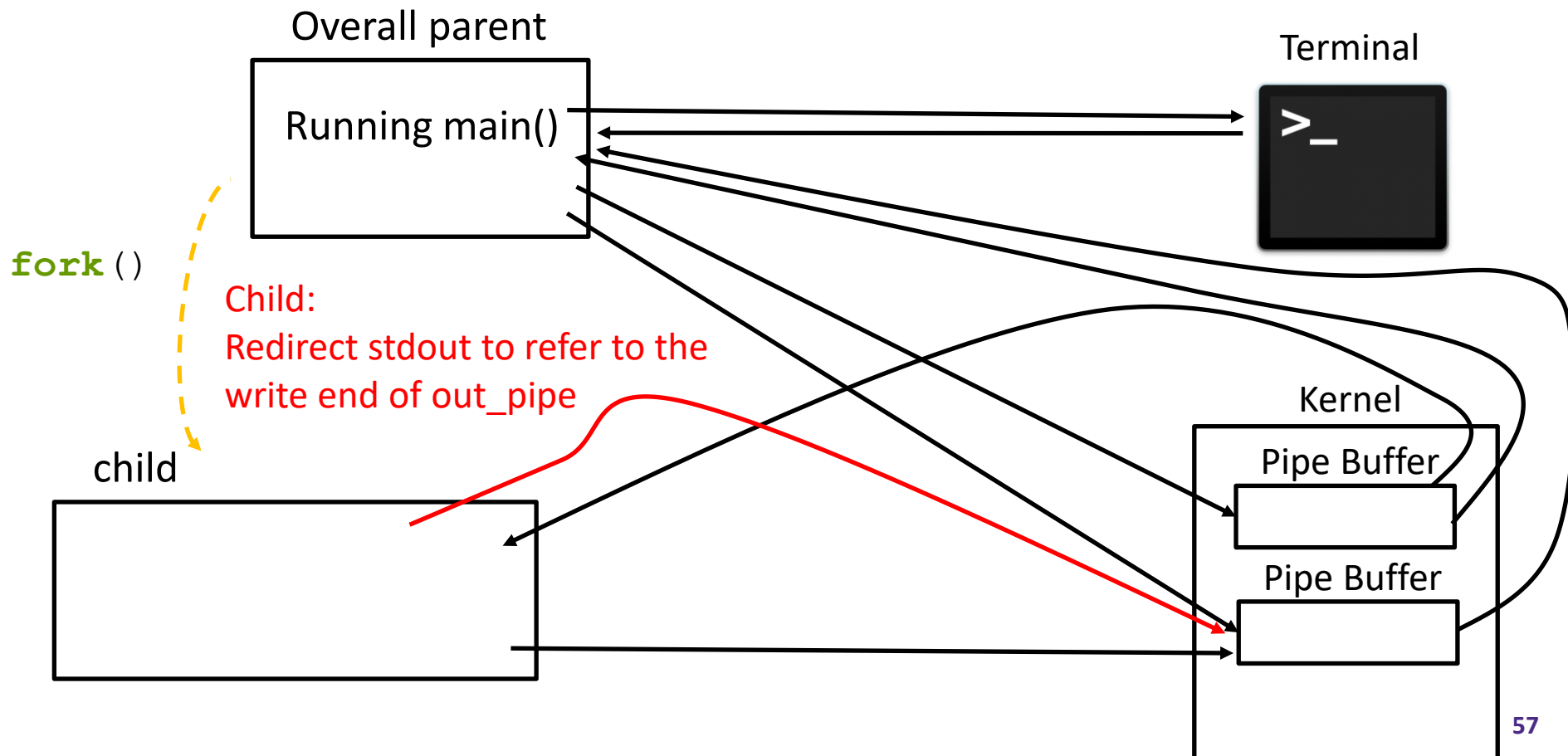
io_autograder.cc Trace

- ❖ Compilation done! Run the compiled program...
BUT send autograder input and capture output



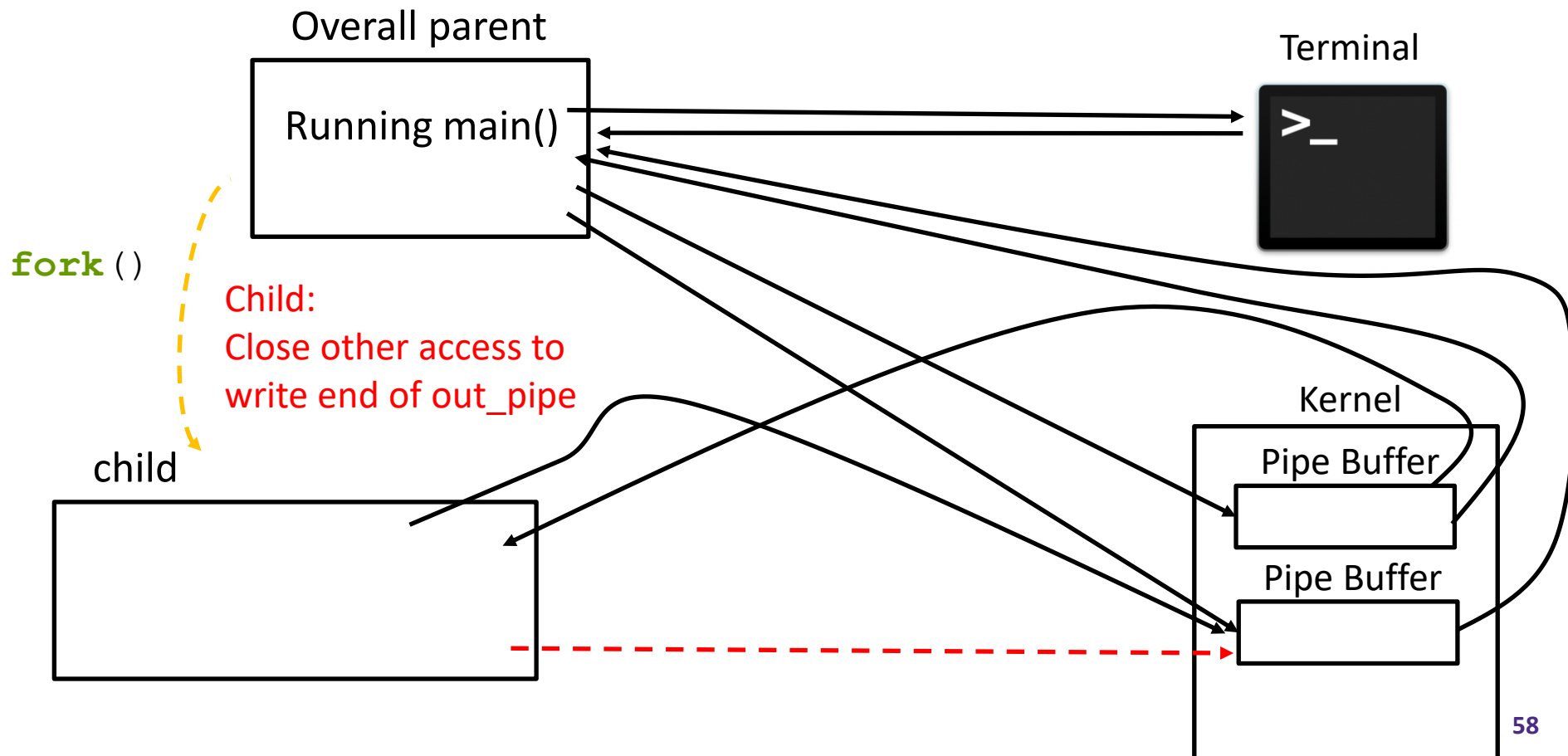
io_autograder.cc Trace

- ❖ Compilation done! Run the compiled program... BUT send autograder input and capture output



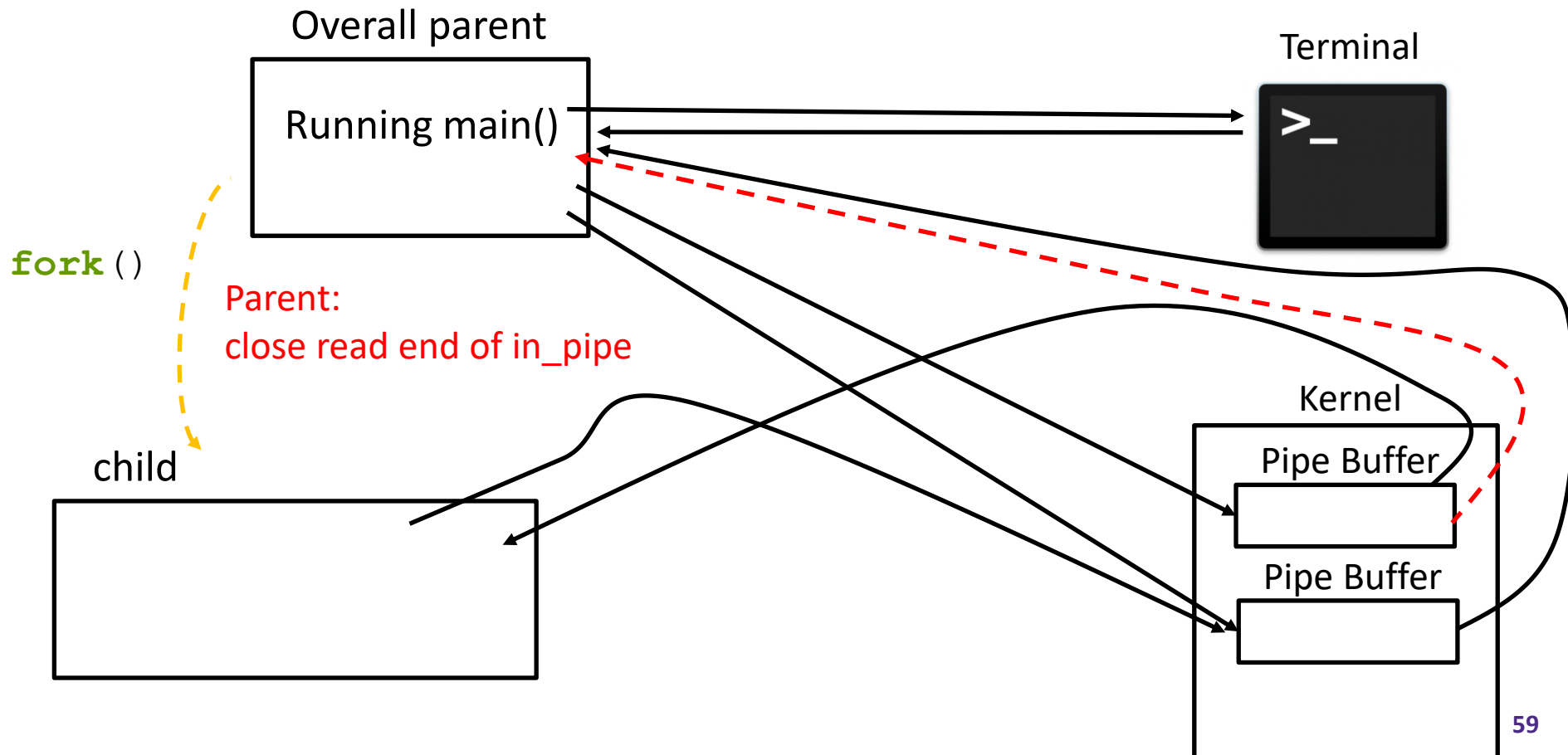
io_autograder.cc Trace

- ❖ Compilation done! Run the compiled program...
BUT send autograder input and capture output



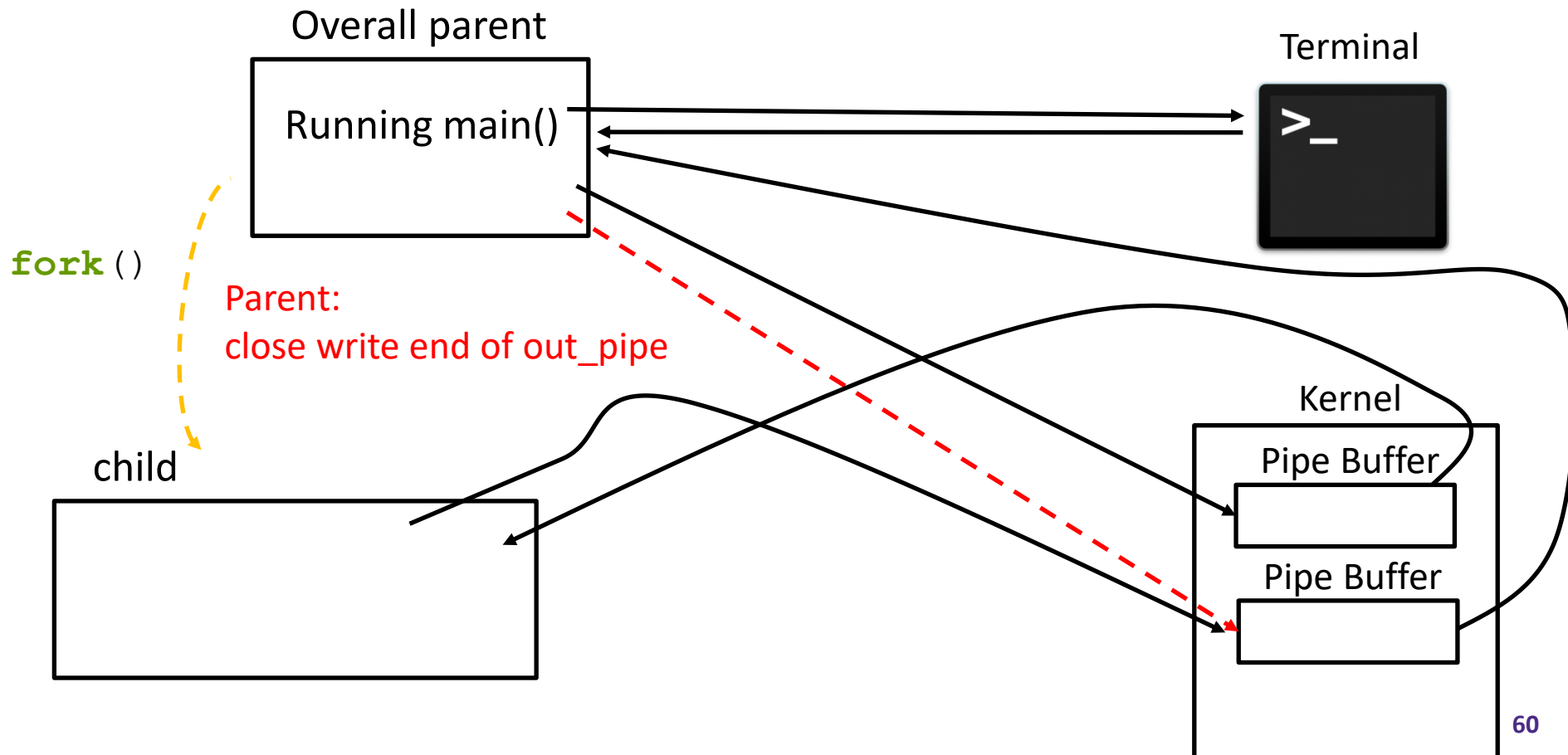
io_autograder.cc Trace

- ❖ Compilation done! Run the compiled program... BUT send autograder input and capture output



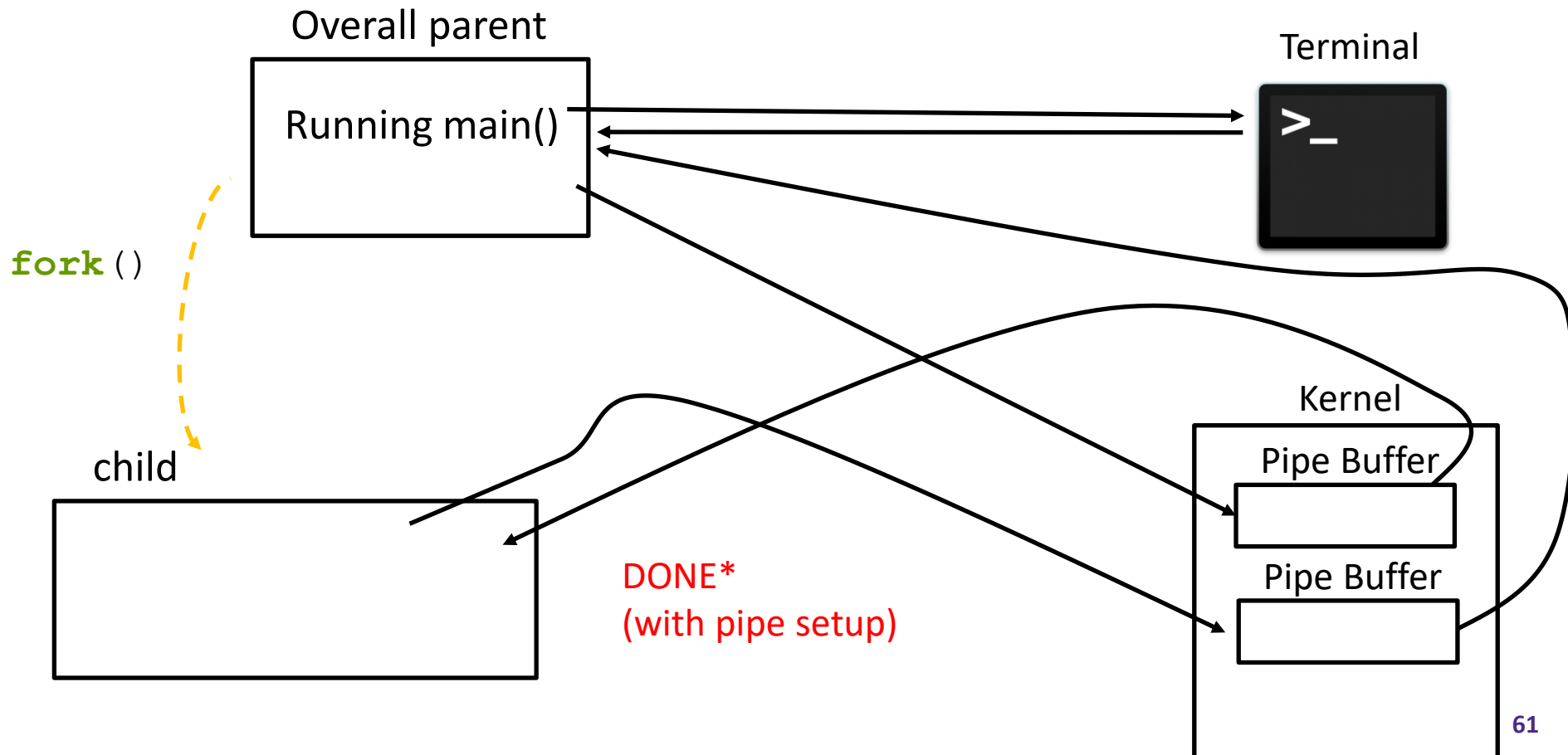
io_autograder.cc Trace

- ❖ Compilation done! Run the compiled program...
BUT send autograder input and capture output



io_autograder.cc Trace

- ❖ Compilation done! Run the compiled program...
BUT send autograder input and capture output



io_autograder.cc Trace

- ❖ Compilation done! Run the compiled program...
BUT send autograder input and capture output

