CS 111 Assignment 3

BYOS (Build your own shell)!

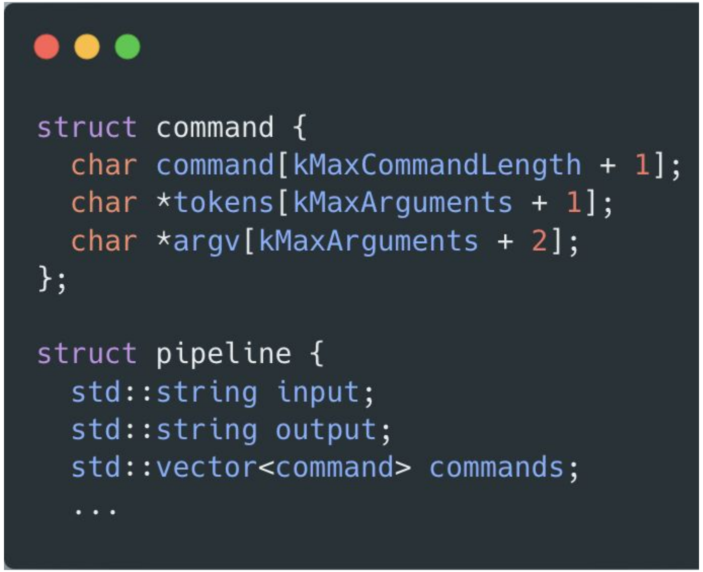
fear the tree

Stanford Shell (`stsh`) Demo

Demo commands:

1. `./samples/stsh_soln`
2. `./samples/stsh_soln < input.txt`

Overview of Parser



```
struct command {
  char command[kMaxCommandLength + 1];
  char *tokens[kMaxArguments + 1];
  char *argv[kMaxArguments + 2];
};

struct pipeline {
  std::string input;
  std::string output;
  std::vector<command> commands;
  ...
```

```
grep abc < stsh.cc | wc -l > out.txt
```

pipeline.input = “stsh.cc”

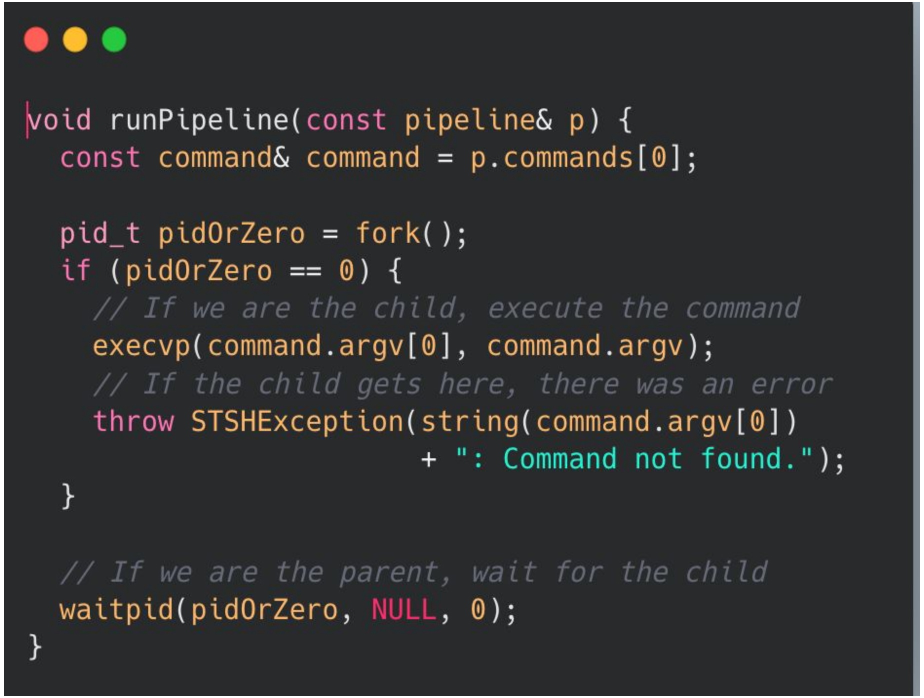
pipeline.output = “out.txt”

pipeline.commands = [com_A, com_B]

com_A: *command* = { .command = “grep”, .tokens = [“abc”], .argv = [“grep”, “abc”] }

com_B: *command* = { .command = “wc”, .tokens = [“-l”], .argv = [“wc”, “-l”] }

Single Commands



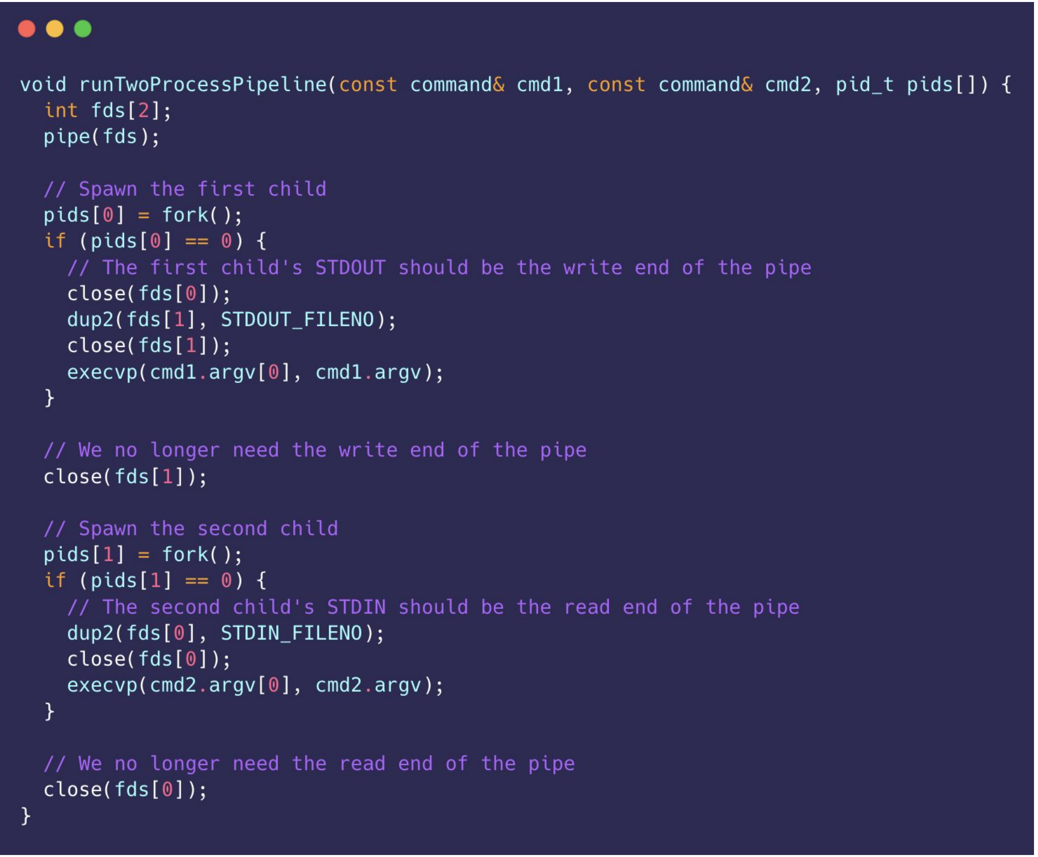
```
void runPipeline(const pipeline& p) {
  const command& command = p.commands[0];

  pid_t pidOrZero = fork();
  if (pidOrZero == 0) {
    // If we are the child, execute the command
    execvp(command.argv[0], command.argv);
    // If the child gets here, there was an error
    throw STSHException(string(command.argv[0])
                        + ": Command not found.");
  }

  // If we are the parent, wait for the child
  waitpid(pidOrZero, NULL, 0);
}
```

Tips:

1. `fork()` returns child's pid to parent and 0 to child
2. Parent should always wait on their children to avoid them becoming zombies
3. `execvp()` starts a new program by wiping the original one, so it never returns if successful
4. Syntax for raise an exception:

```
throw SOME_EXCEPTION(err_msg)
```

Two Processes Pipeline



```
void runTwoProcessPipeline(const command& cmd1, const command& cmd2, pid_t pids[]) {
  int fds[2];
  pipe(fds);

  // Spawn the first child
  pids[0] = fork();
  if (pids[0] == 0) {
    // The first child's STDOUT should be the write end of the pipe
    close(fds[0]);
    dup2(fds[1], STDOUT_FILENO);
    close(fds[1]);
    execvp(cmd1.argv[0], cmd1.argv);
  }

  // We no longer need the write end of the pipe
  close(fds[1]);

  // Spawn the second child
  pids[1] = fork();
  if (pids[1] == 0) {
    // The second child's STDIN should be the read end of the pipe
    dup2(fds[0], STDIN_FILENO);
    close(fds[0]);
    execvp(cmd2.argv[0], cmd2.argv);
  }

  // We no longer need the read end of the pipe
  close(fds[0]);
}
```

note: you should call `waitpid` at the end

1. These two child processes should run *simultaneously* (e.g. `sleep 2 | sleep 3` will wait for ~3 seconds, not 5).
2. Remember to close unused file descriptors ("FDs").
3. `dup2` is very useful! You can duplicate a FD to whichever number you like.
4. You can use `pipe2` with `O_CLOEXEC` instead of `pipe` to save yourself some `close` calls.
5. Recall that children inherit copies of their parent's FDs.

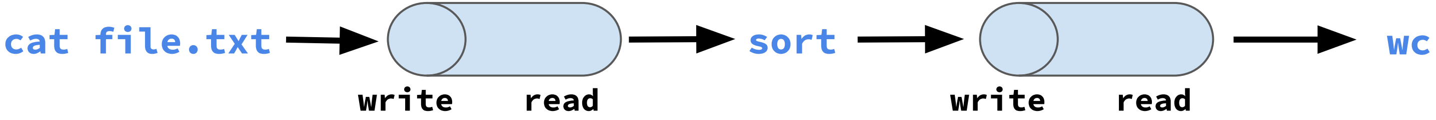
Arbitrarily long pipelines

Pipeline of **more than two** processes

```
cat file.txt | sort | wc
```

- ➔ The **output** of `cat file.txt` becomes the **input** of `sort`
- ➔ The **output** of `sort` becomes the **input** of `wc`
- ➔ **N** processes and **N - 1** pipes
- ➔ The **first** program only has its **STDOUT** redirected
- ➔ The **last** program only has its **STDIN** redirected

At this point, you should strive to *generalize* your previous 2-process pipeline solution!



Credit: CS110 slides by Sophie Decoppet

Input and Output Redirection

- Input redirection: redirect STDIN to read from an existing file
- Output redirection: redirect STDOUT to write to a (possibly existing) file

```
cat < inputFile.txt | wc > outputFile.txt
```

Credit: CS110 slides by Sophie Decoppet

Input and Output Redirection

- Input redirection: redirect STDIN to read from an existing file
- Output redirection: redirect STDOUT to write to a (possibly existing) file

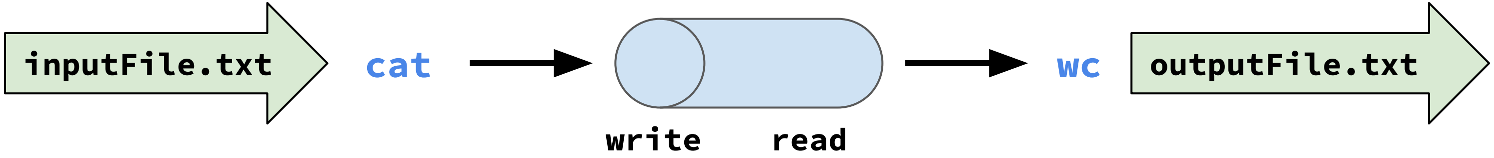
$$\texttt{cat < }\underbrace{\texttt{inputFile.txt}}_{\text{Input file}}\texttt{ | wc > }\underbrace{\texttt{outputFile.txt}}_{\text{output file}}$$

Credit: CS110 slides by Sophie Decoppet

Input and Output Redirection

- Input redirection: redirect STDIN to read from an existing file
- Output redirection: redirect STDOUT to write to a (possibly existing) file

```
cat < inputFile.txt | wc > outputFile.txt
```

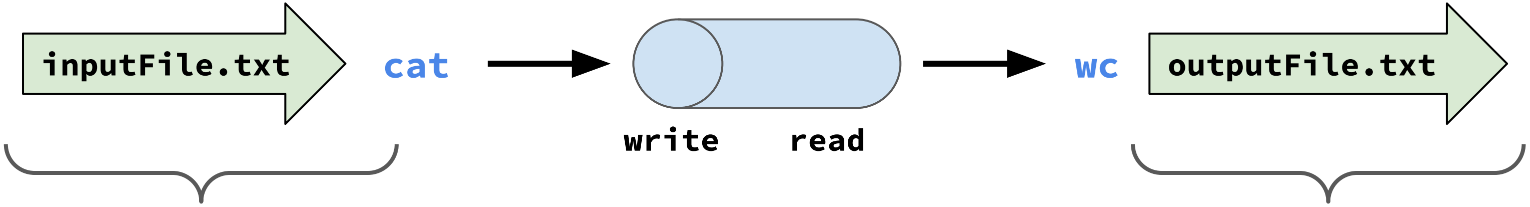


Credit: CS110 slides by Sophie Decoppet

Input and Output Redirection

- Input redirection: redirect STDIN to read from an existing file
- Output redirection: redirect STDOUT to write to a (possibly existing) file

```
cat < inputFile.txt | wc > outputFile.txt
```



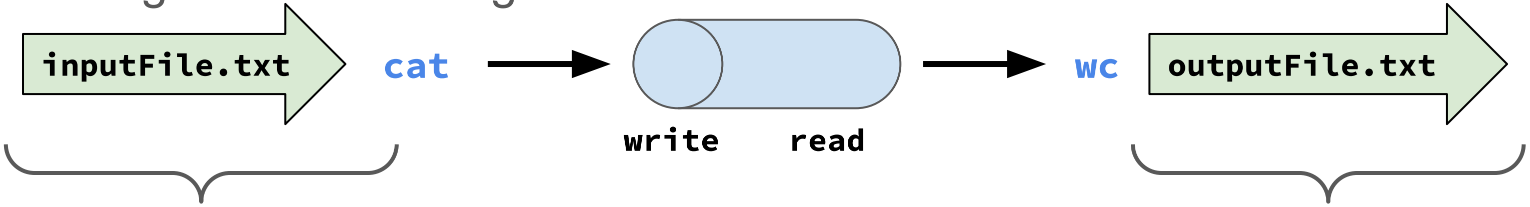
Open this file using **open()**

Open or create(!) this file using **open()**

Credit: CS110 slides by Sophie Decoppet

Input and Output Redirection

- Hint: Only the STDIN of the first process and/or the STDOUT of the last process will ever change because of I/O redirection.
- Hint #2: Once you've opened the input and/or output files appropriately, consider how we can leverage what we know about FDs to redirect input or output to an open file.
- Don't forget error handling!



Open this file using **open()**

Open or create(!) this file using **open()**

Credit: CS110 slides by Sophie Decoppet

Testing - sanity check is not exhaustive!

Good Start: Short test programs

- `conduit`: reads one character from standard input every second and (after a possible delay) publishes one or more copies of that letter
- `spin`: spins for n seconds
- `sigsegv`: spins for n seconds and then raise SIGSEGV.
- `split`: forks and waits for a child which spins for n seconds
- `open_fds`: prints **its** currently open file descriptors

Use provided reference solution

```
./samples/stsh-soln
```

Debugging

GDB

You will need to run some special commands to use GDB with `stsh`. Please refer to the assignment specification for the juicy details.

Valgrind

You can use Valgrind to track open file descriptors with `valgrind --track-fds=yes ./stsh` although it is not supported for debugging memory leaks or errors on this assignment.

`inspect-fds.py`

If you log into the same Myth machine from another SSH session, you can run `./samples/inspect-fds.py stsh` to see all the file descriptors in use by `stsh` (or any program you pass in).

Print statements (this one speaks for itself)

Not to *bash* your shell too much, but `stsh` >> everything else

Any questions?