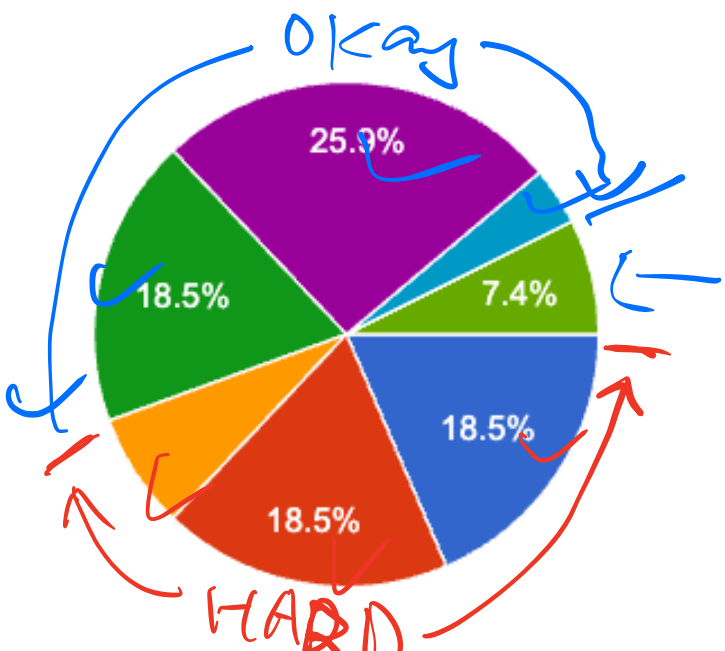
Choose your feeling about Lab1:

27 responses

Copy



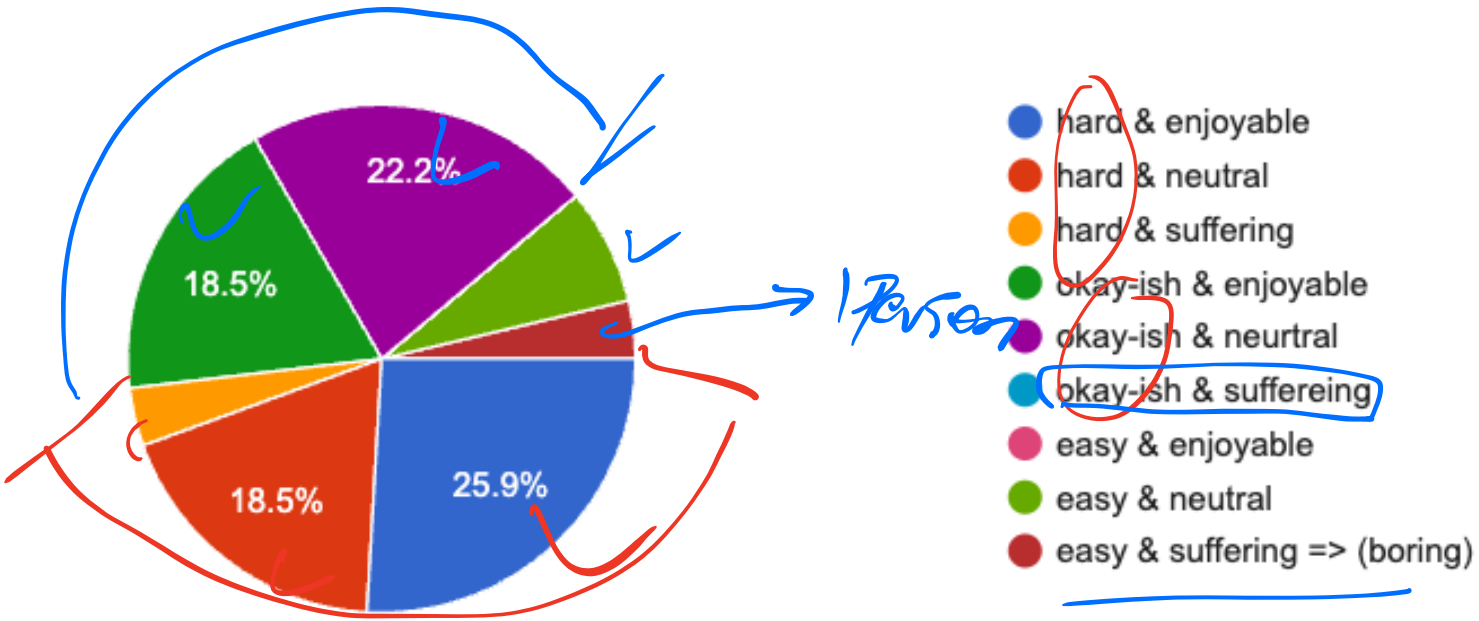
difficulty feeling

- hard & enjoyable
- hard & neutral
- hard & suffering
- okay-ish & enjoyable
- okay-ish & neurtral
- okay-ish & suffereing
- easy & enjoyable
- easy & neutral
- easy & suffering => (boring)

Choose your current feeling about CS5600:

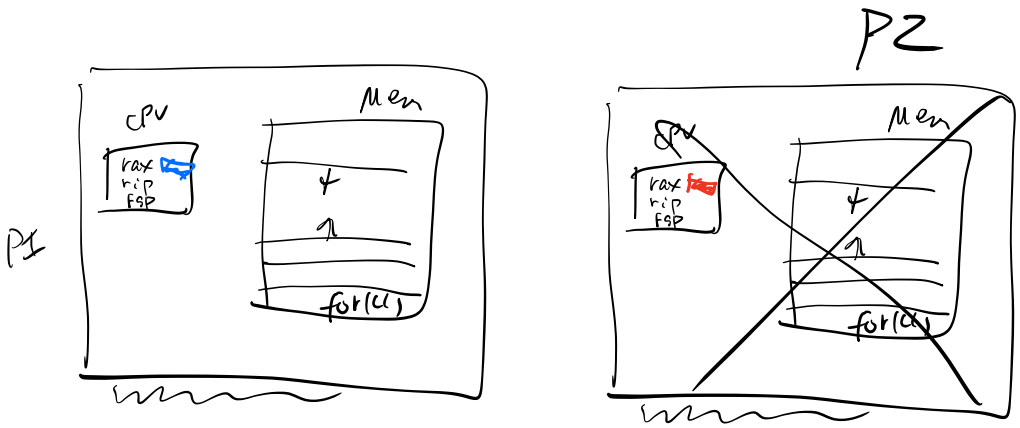
Copy

27 responses



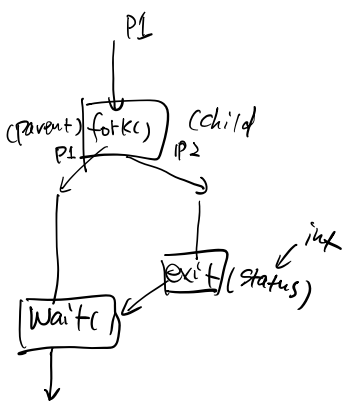
Week 3.b
CS5600
1/25 2023
https://naizhengtan.github.io/23spring/

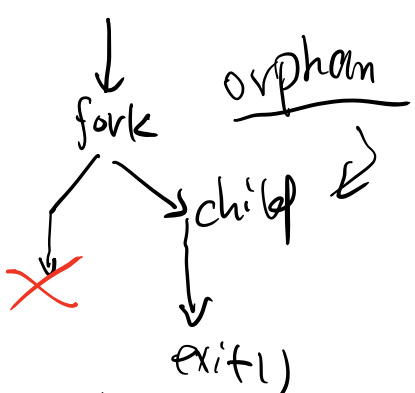
1. Survey
2. Process birth
3. Shell crash course
4. Shell internals I
5. File descriptors
6. Shell internals II



Sys Call

fork()

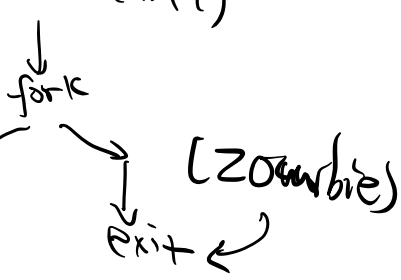


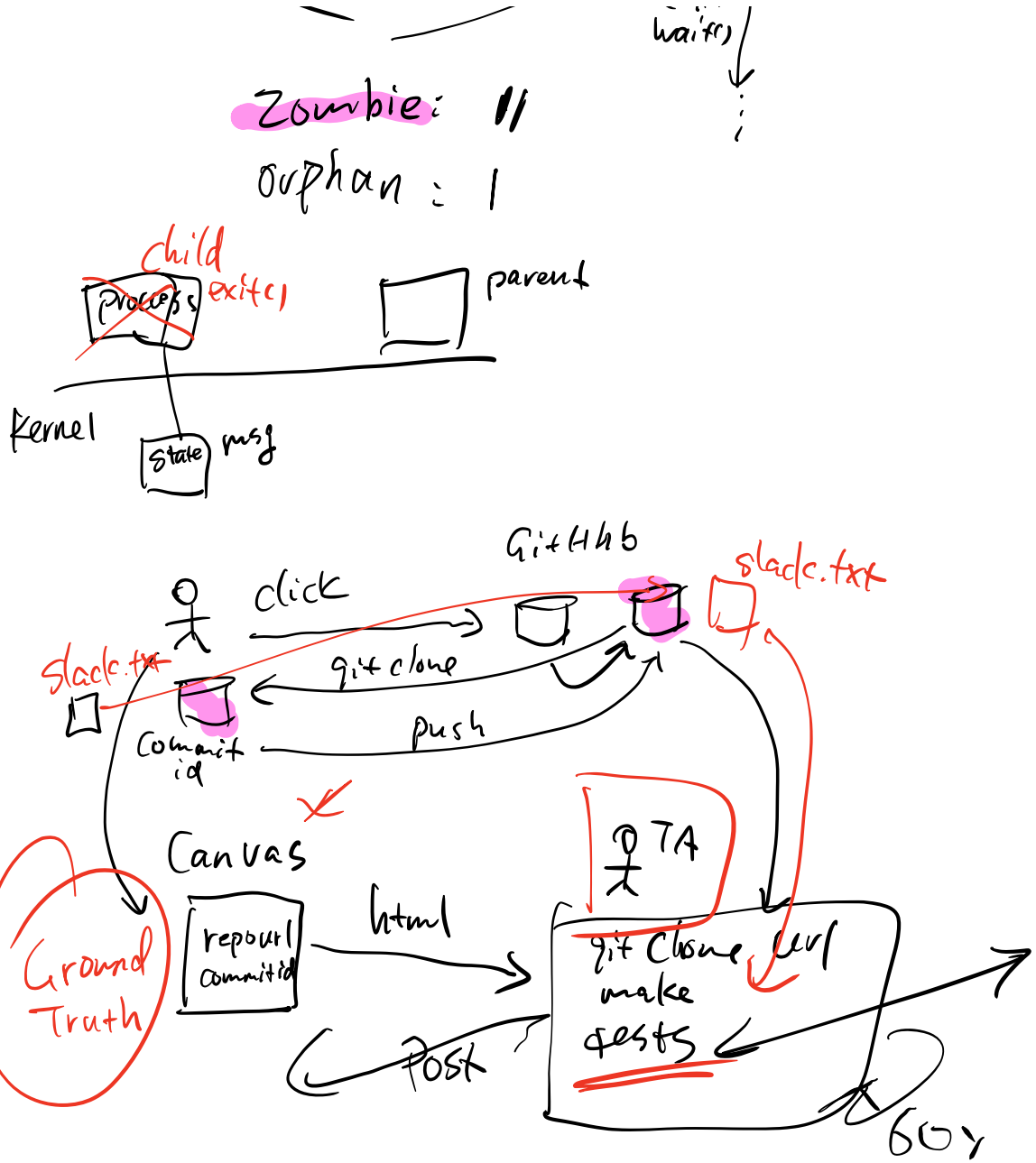


pid ← ret

```
if (fork() == 0) {
    // child
} else {
    // parent
}
```

Q: orphan vs. zombie harmful?





Slack hours. 120hr. Labs

50% ——— Q: ——→ 60hrs late

60hr slack

Crash course, Shell ← bash/zsh GUI

1. run cmd
   "$ ls" and "$ ls -a"

2. output redirection
   "$ ls" prints to screen
   "$ ls > files.txt"
   $

3. backgrounding
   "$ web-server &
   $ "

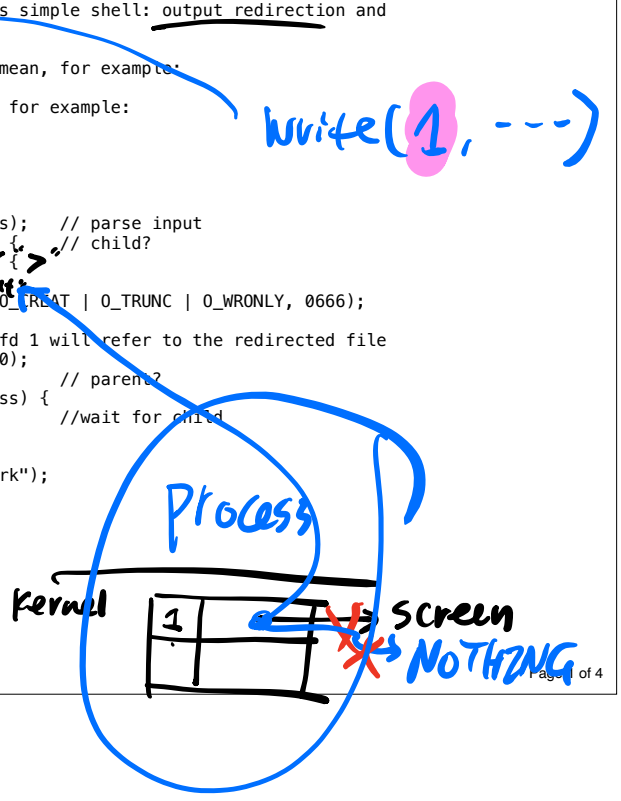
while (true) {
  accept_req();
  handle_req();
}

4. pipe
   -- "$ cat students.txt | shuf -n 1"
   -- equivalent to
      "$ cat students.txt > /tmp/tmpfile
       $ shuf -n 1 /tmp/tmpfile
       $ rm /tmp/tmpfile"

$ [Cmd]

5. Shell builtin cmds vs. program
   -- "echo/pwd/which" vs. "ls"
   -- use "which" to tell
      program: "$ which ls"
                  => "/bin/ls" ← path
      built-in: "$ which which"
                  => "which: shell built-in command"

```
CS5600 23spring
Handout week03b

The handout is meant to:

--illustrate how the shell itself uses syscalls

--communicate the power of the fork()/exec() separation

--give an example of how small, modular pieces (file descriptors,
pipes, fork(), exec()) can be combined to achieve complex behavior
far beyond what any single application designer could or would have
specified at design time.

1. Pseudocode for a very simple shell

    while (1) {
      write(1, "$ ", 2);
      readcommand(command, args); // parse input
      if ((pid = fork()) == 0) {  // child?
      execve(command, args, 0);
      } else if (pid > 0) {       // parent?
        wait(0);                  //wait for child
      } else {
        perror("failed to fork");
    }}

2. Now add two features to this simple shell: output redirection and
   backgrounding

    By output redirection, we mean, for example:
        $ ls > list.txt
    By backgrounding, we mean, for example:
        $ myprog &
        $

    while (1) {
      write(1, "$ ", 2);
      readcommand(command, args);   // parse input
      if ((pid = fork()) == 0) {    // child?
        if (output_redirected) {
        close(1);
        open(redirect_file, O_CREAT | O_TRUNC | O_WRONLY, 0666);
        }
        // when command runs, fd 1 will refer to the redirected file
        execve(command, args, 0);
      } else if (pid > 0) {       // parent?
        if (foreground_process) {
          wait(0);                //wait for child
        }
      } else {
        perror("failed to fork");
      }
    }
```




```
3. Another syscall example: pipe()

  The pipe() syscall is used by the shell to implement pipelines, such as
      $ ls | sort | head -4
  We will see this in a moment; for now, here is an example use of
  pipes.

  // C fragment with simple use of pipes

  int fdarray[2];
  char buf[512];
  int n;

  pipe(fdarray);
  write(fdarray[1], "hello", 5);
  n = read(fdarray[0], buf, sizeof(buf));
  // buf[] now contains 'h', 'e', 'l', 'l', 'o'

4. File descriptors are inherited across fork

  // C fragment showing how two processes can communicate over a pipe

  int fdarray[2];
  char buf[512];
  int n, pid;

  pipe(fdarray);
  pid = fork();
  if(pid > 0){
    write(fdarray[1], "hello", 5);
  } else {
    n = read(fdarray[0], buf, sizeof(buf));
  }
```

Motivation of redirection & pipe.

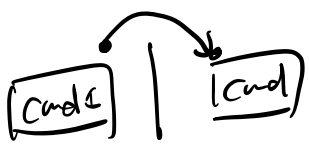
Q: 2 txt file ⟹ new file ?
concat

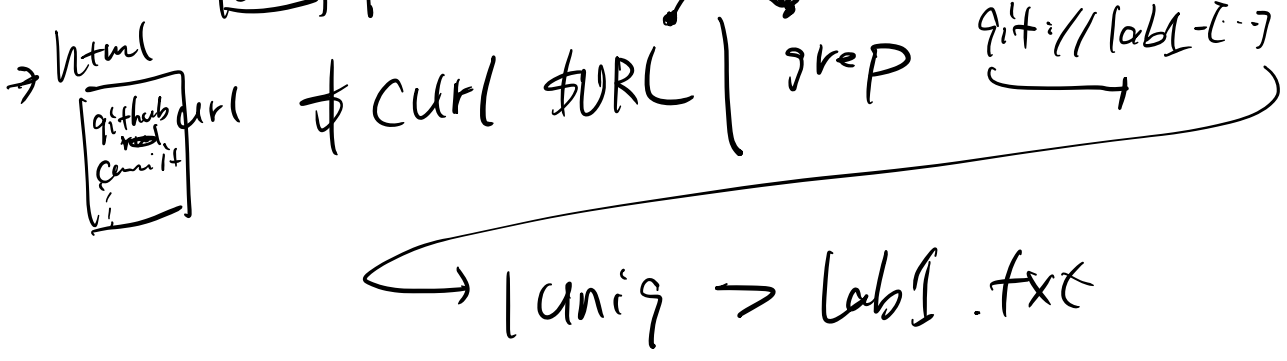
10 txt files

10000

$ cat file1.txt file2.txt > new_file

pipe





File descriptor:

process int fd = open("/tmp/xyz"...)
3

3: 0: stdin
1: stdout ← "hello world!"
2: stderr ← "cat 02: ..."