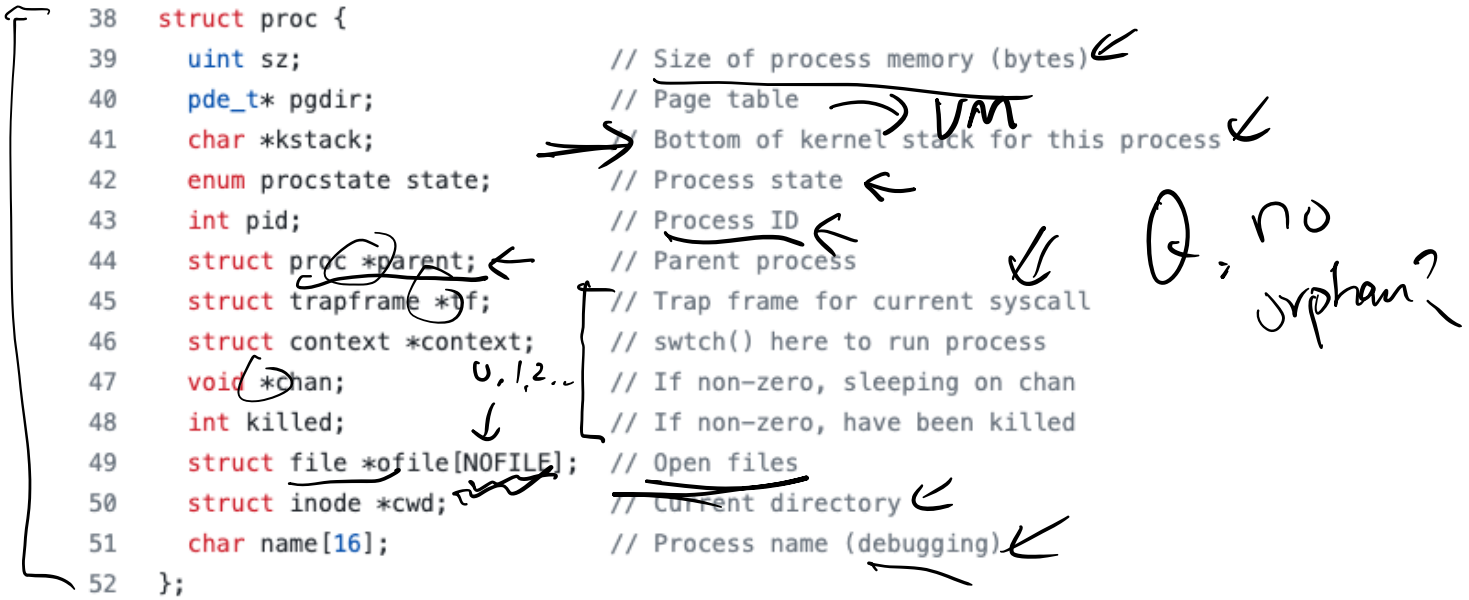
```
36
37  // Per-process state
38  struct proc {
39    uint sz;                     // Size of process memory (bytes)
40    pde_t* pgdir;                // Page table
41    char *kstack;                // Bottom of kernel stack for this process
42    enum procstate state;        // Process state
43    int pid;                     // Process ID
44    struct proc *parent;         // Parent process
45    struct trapframe *tf;        // Trap frame for current syscall
46    struct context *context;     // swtch() here to run process
47    void *chan;                  // If non-zero, sleeping on chan
48    int killed;                  // If non-zero, have been killed
49    struct file *ofile[NOFILE];  // Open files
50    struct inode *cwd;           // Current directory
51    char name[16];               // Process name (debugging)
52  };
53
```



Borrowed from xv6: https://github.com/mit-pdos/xv6-public/blob/eeb7b415dbcb12cc362d0783e41c3d1f44066b17/proc.h

Week 4.a
CS5600
1/30 2023
https://naizhengtan.github.io/23spring/

1. Shell internal continued & discussions
2. Implementation of processes
3. Context switch intro
4. Scheduling intro

02/05 , 02/17

Q: fork() ret val? ← child ret = 0 parent ret ← pid of child

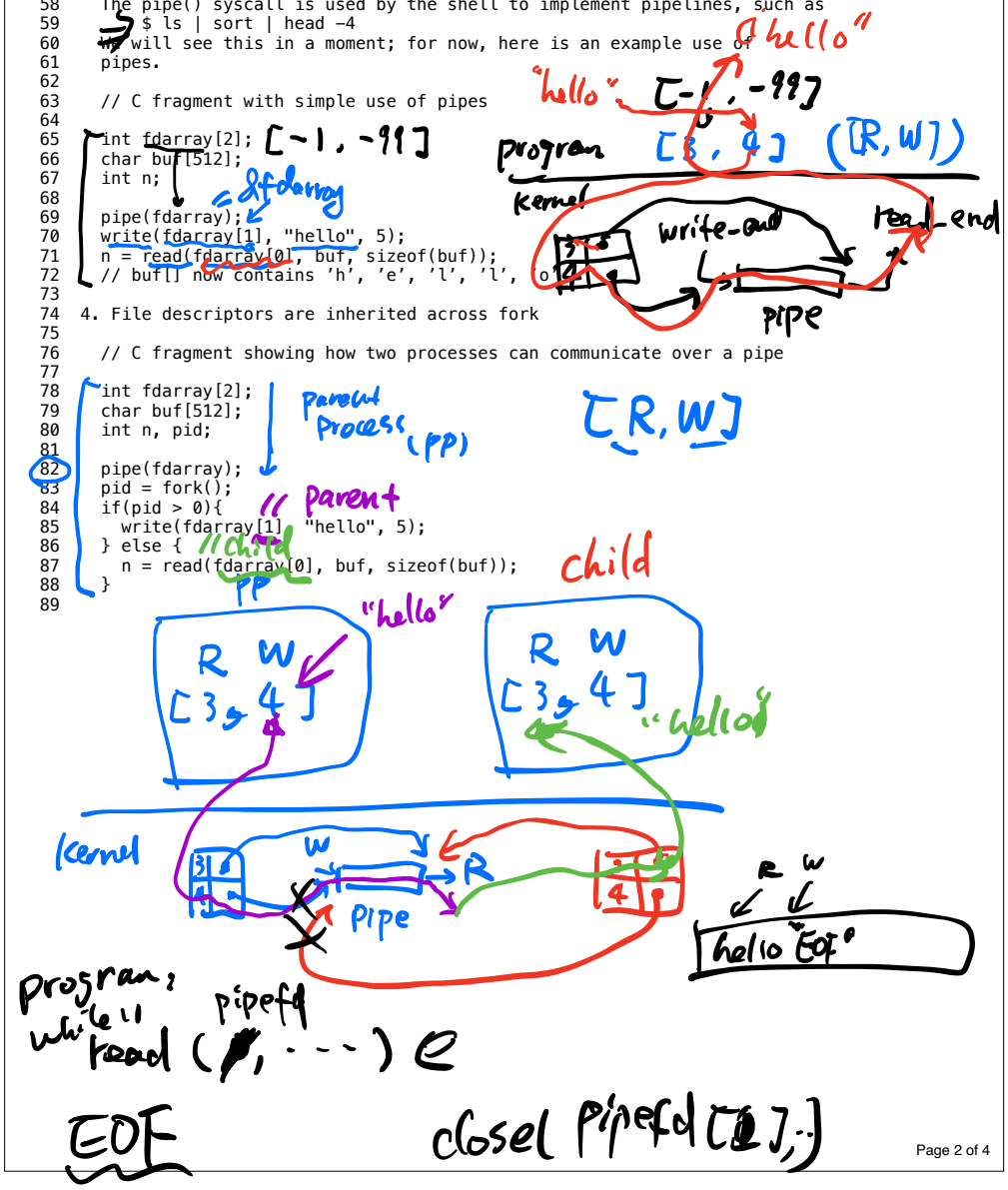
Q: fd 0/1/2? ← 0: stdin 1: stdout 2: stderr

Q: "ls > log.txt"?

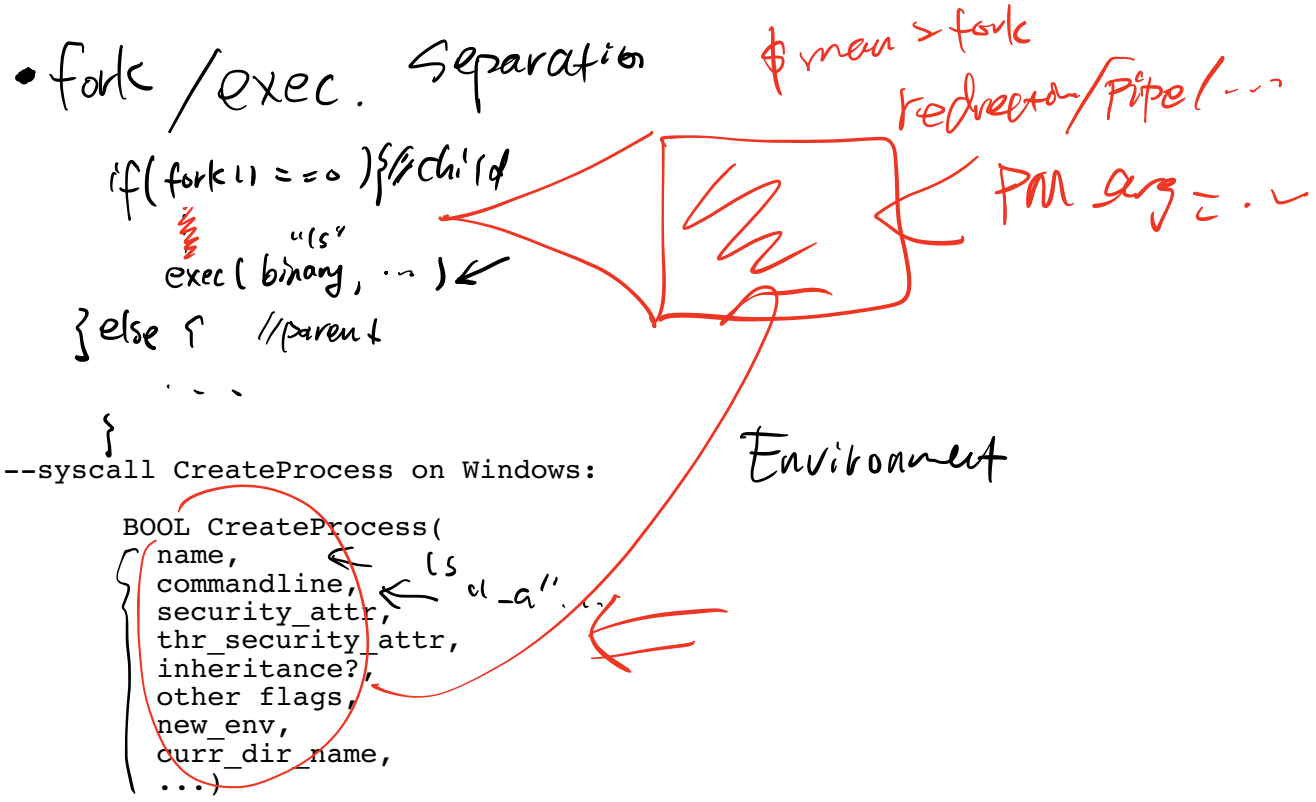
```
CS5600 23spring
Handout week03b

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));
  }
```



- fork/exec. Separation



- good abstraction?
  - fd ← open/close/read/write
  - fd 0/1/2
  - fork/exec.
- "A fork() in the road"
  HotOS'19

"Fork today is a convenient API for a single-threaded process with a small memory footprint and simple memory layout that requires fine-grained control over the execution environment of its children but does not need to be strongly isolated from them. In other words, a shell."

——"A fork() in the road"

Security

Virtual/Memory

"Fork doesn't Compose"

— example:

```
print("hello world");
fork();
print("\n");
```

2 processes

QUESTION: what do you expect to see on screen?

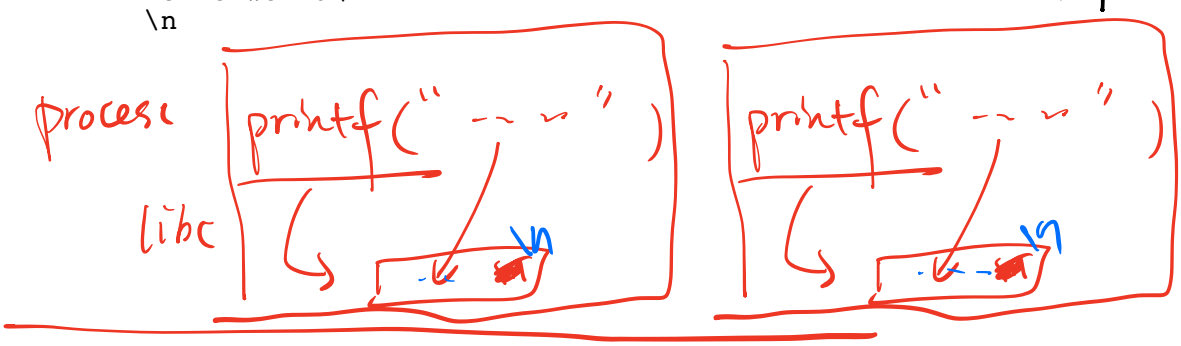
3

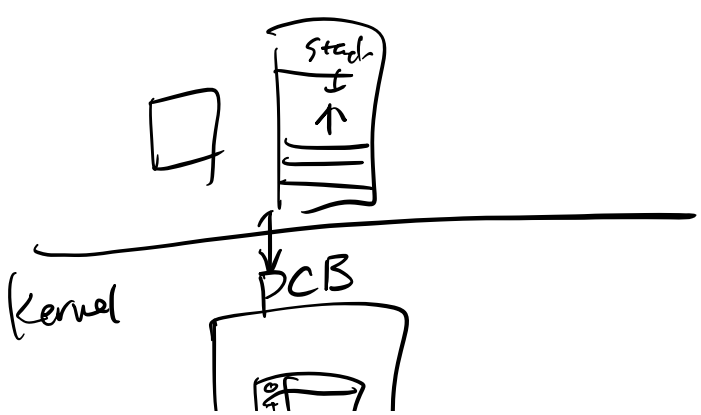
A:
```
hello world\n
hello world\n
```

B:
```
hello world\n
\n
```

A or B

111





- Context Switch