Operating Systems Practice

Operating System Interface

Eunji Lee

(ejlee@ssu.ac.kr)

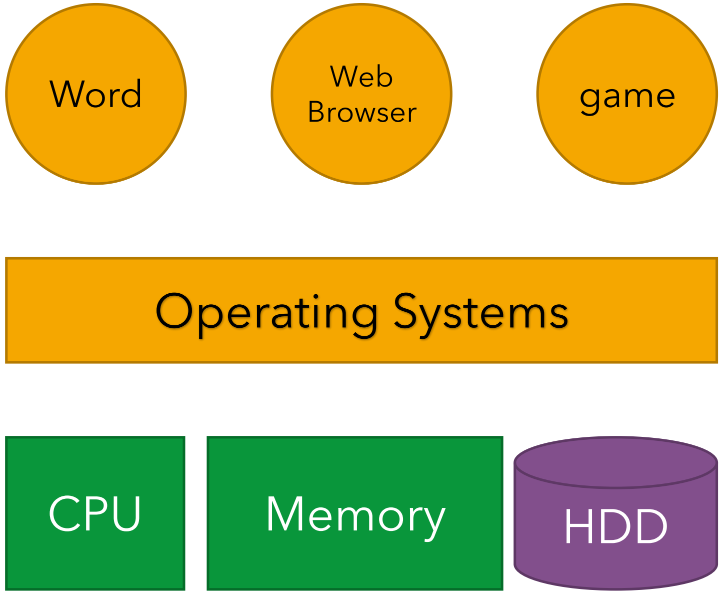
References

- Xv6-books
  - https://pdos.csail.mit.edu/6.828/2018/xv6/book-rev10.pdf
- Contents
  - **Ch.0: Operating system interface**
  - Ch.1: Operating system organization
  - Ch.2: Page tables
  - Ch.3: Traps, interrupts, and drivers
  - Ch.4: Locking
  - Ch.5: Scheduling
  - Ch.6: File system
  - Ch.7: Summary
  - Appendix A: PC hardware
  - Appendix B: The boot loader

Job of Operating Systems

- **Share** a computer among multiple programs
- Manage and ***abstract*** the low-level hardware
- Run multiple programs at the same time

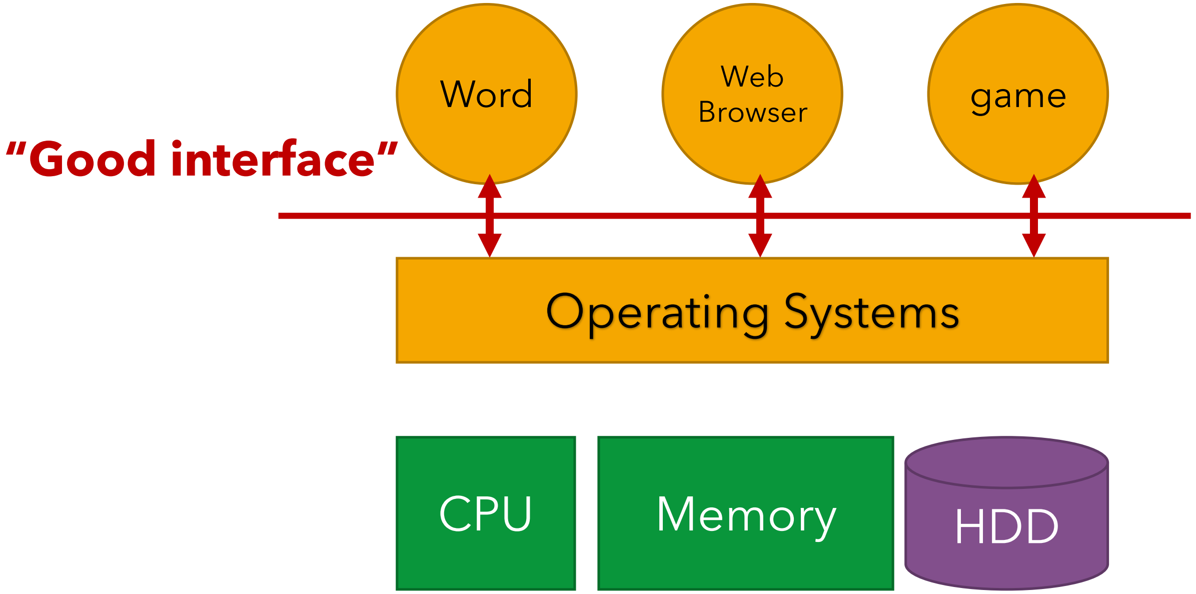
- Key ingredient: "Good interface"



Operating System Interfaces

- Requirements
  - Easy-to-use
  - Sophisticated features

- How to design interface?
  - Rely on a few mechanisms
  - Combined to provide generality



System call

- Operating system interface
- Invoked when a process needs to invoke a kernel service
- CPU's hardware protection
  - Ensure each process in user space can access only its own memory
  - When a user process invokes a system call, the hardware raises the privilege level and starts executing a pre-arranged function (system call procedures) in the kernel
- Services: Processes, memory, file system, pipes, etc.

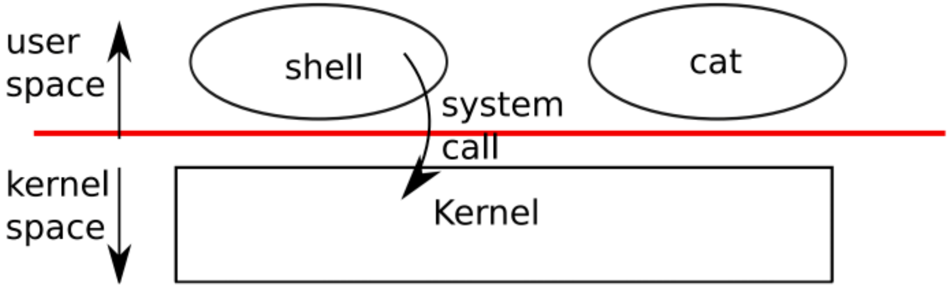
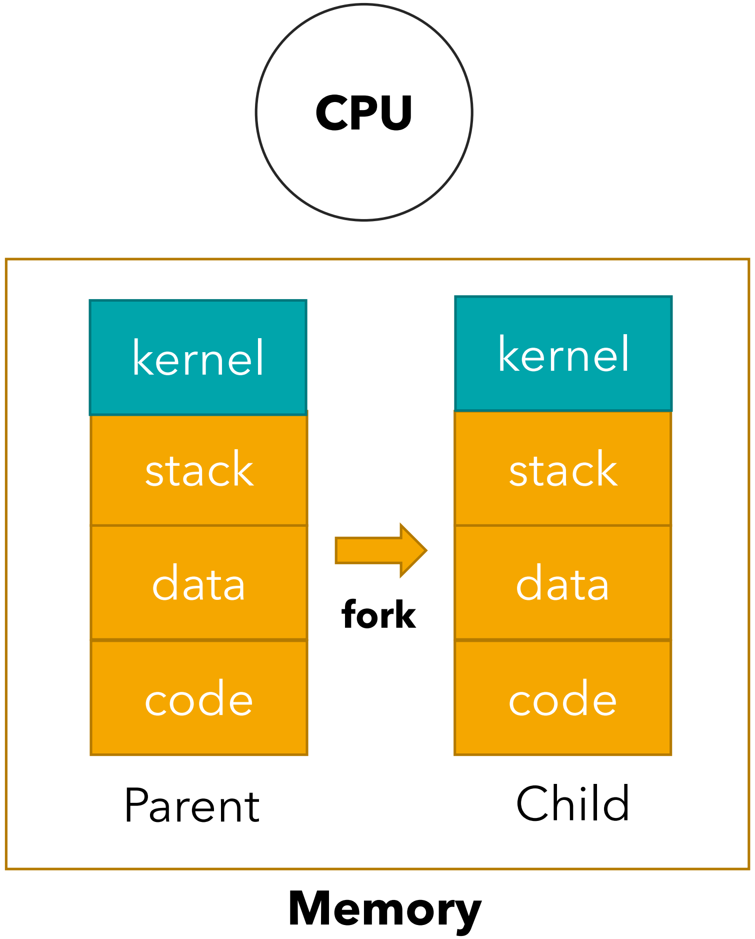


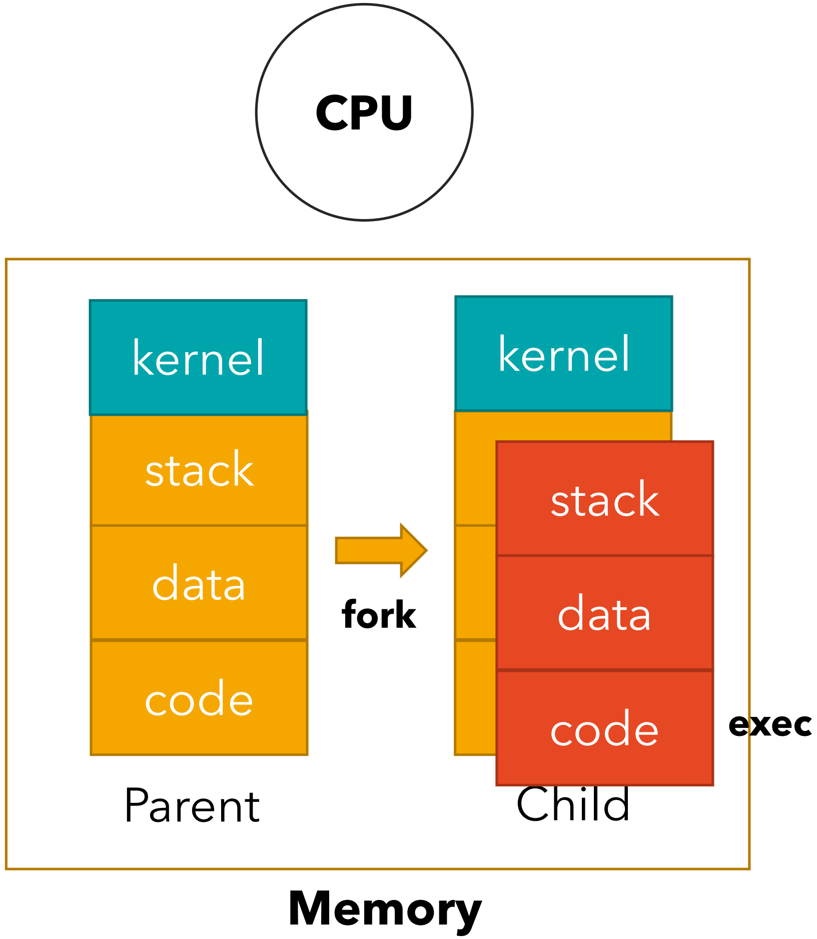
Figure 0-1. A kernel and two user processes.

Processes and memory



```
// fork.c
int main()
{
    int pid = fork();

    if (pid > 0) {
        printf(“parent: child=%d\n“, pid);
        pid = wait();
        printf(“child %d is done\n“, pid);
    } else if (pid == 0) {
        printf(“child: exiting\n");
    } else{
        printf(“fork error\n");
    }
    return 0;
}
```

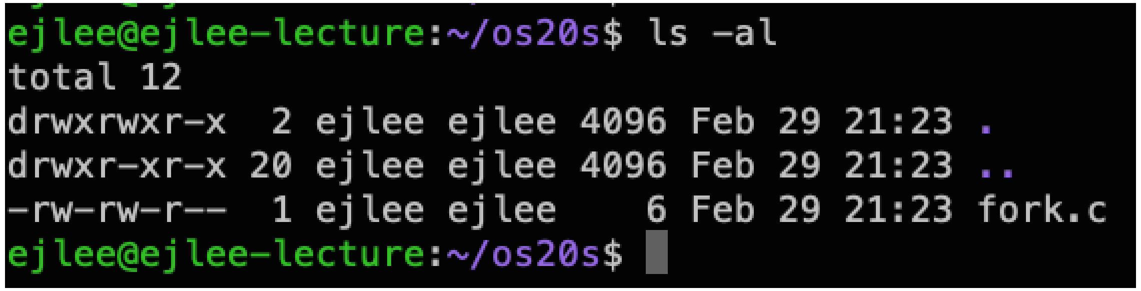
Processes and memory



```
// exec.c
int main()
{
    char *argv[3];
    argv[0] = “echo”
    argv[1] = “hello”
    argv[2] = 0;

    int pid = fork();
    ...
    } else if (pid == 0) {
        printf(“child: exiting\n");

        execve(“/bin/echo”, argv);
    }
    ...
}
```

Processes and memory

- Shell

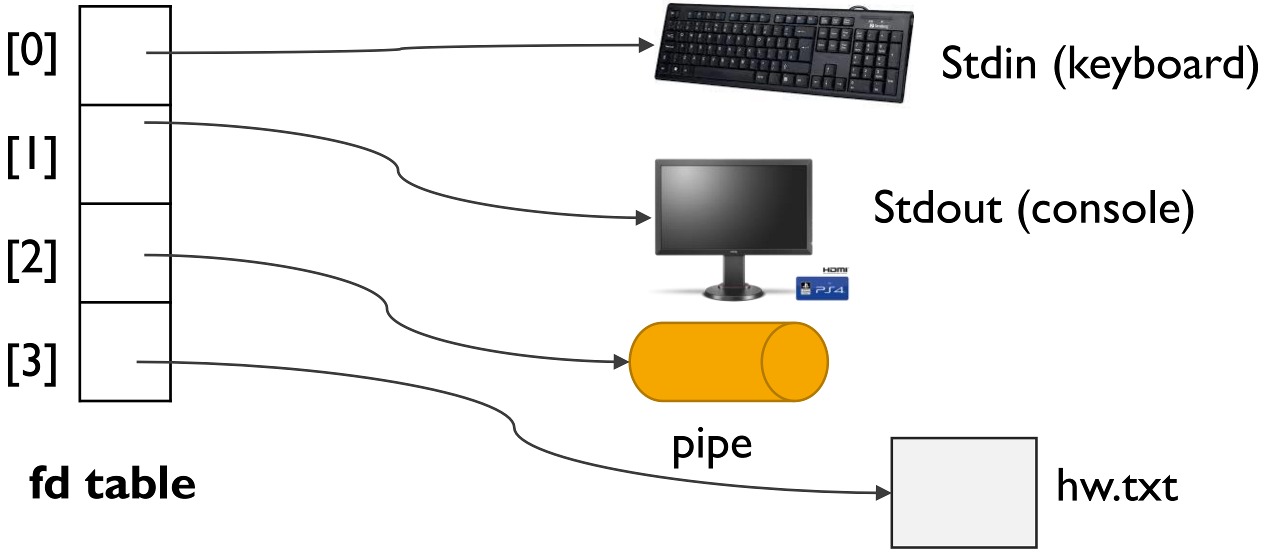


```
ejlee@ejlee-lecture:~/os20s$ ls -al
total 12
drwxrwxr-x  2 ejlee ejlee 4096 Feb 29 21:23 .
drwxr-xr-x 20 ejlee ejlee 4096 Feb 29 21:23 ..
-rw-rw-r--  1 ejlee ejlee    6 Feb 29 21:23 fork.c
ejlee@ejlee-lecture:~/os20s$
```

```
// sh.c

int main()
{
    while(1) {
      printf(“$ ”);
      getcmd(cmd);

      int pid = fork();
      if(pid > 0) {
          wait()
      } else if (pid == 0) {
          exec(cmd);
      }
   }
}
```

I/O and File descriptors

- File descriptor
  - A small integer representing a kernel object that a process may read from or write to
  - Abstract away the differences between files, pipes, and devices
  - Make them all look lie streams of bytes
  - Kernel uses a file descriptor as an index into a per-process table



[0]

[1]

[2]

[3]

fd table

Stdin (keyboard)

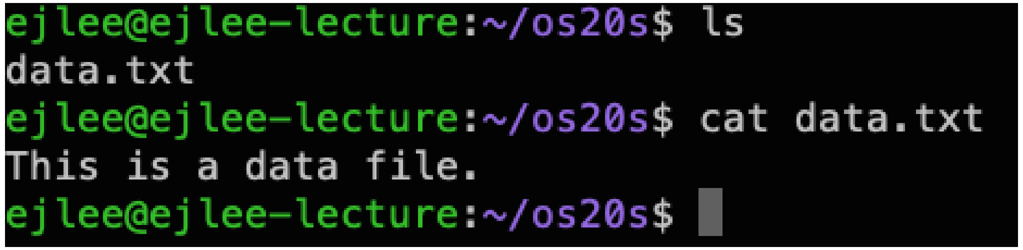
Stdout (console)

pipe

hw.txt

I/O and File descriptors

- cat



```
ejlee@ejlee-lecture:~/os20s$ ls
data.txt
ejlee@ejlee-lecture:~/os20s$ cat data.txt
This is a data file.
ejlee@ejlee-lecture:~/os20s$ 
```

I/O and File descriptors

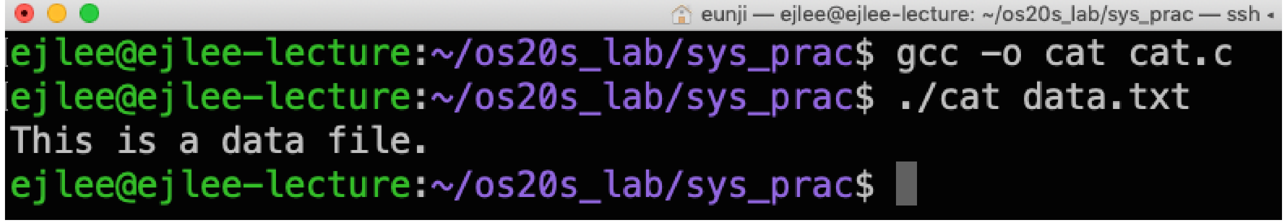
- cat

```
// cat.c
void cat(int fd)
{
  int n;

  while((n = read(fd, buf, sizeof(buf))) > 0) {
    if (write(stdout, buf, n) != n) {
      printf("cat: write error\n");
      return;
    }
  }
  if(n < 0){
    printf("cat: read error\n");
    return;
  }
}
```

```
int main(int argc, char *argv[])
{
  int fd, i;

  if(argc <= 1){
    cat(0);
    exit();
  }

  for(i = 1; i < argc; i++){
    if((fd = open(argv[i], 0)) < 0){
      printf(1, "cat: cannot open %s\n", argv[i]);
      exit();
    }
    cat(fd);
    close(fd);
  }
  exit();
}
```

I/O and File descriptors

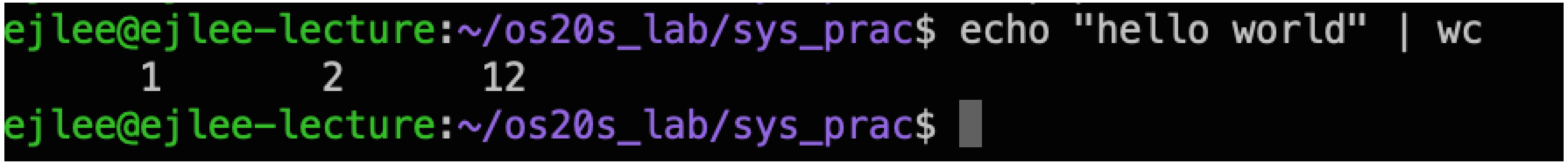
- cat



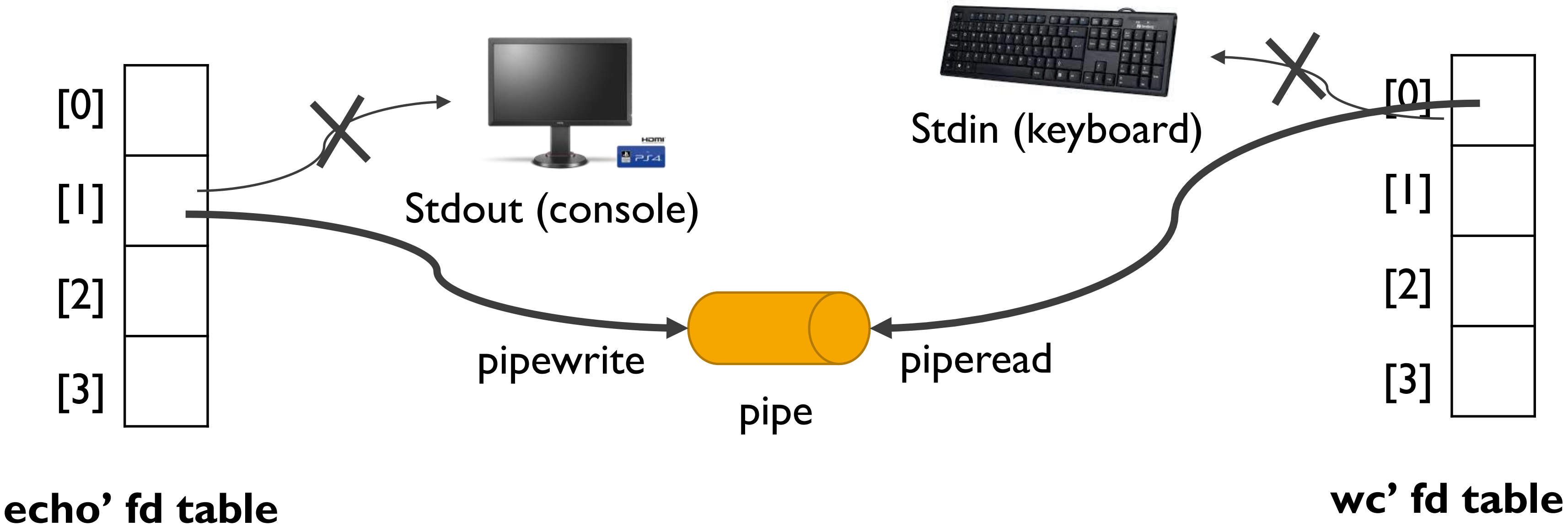
```
ejlee@ejlee-lecture:~/os20s_lab/sys_prac$ gcc -o cat cat.c
ejlee@ejlee-lecture:~/os20s_lab/sys_prac$ ./cat data.txt
This is a data file.
ejlee@ejlee-lecture:~/os20s_lab/sys_prac$ 
```

Pipes

- A small kernel buffer exposed to processes as a pair of file descriptors
- Example

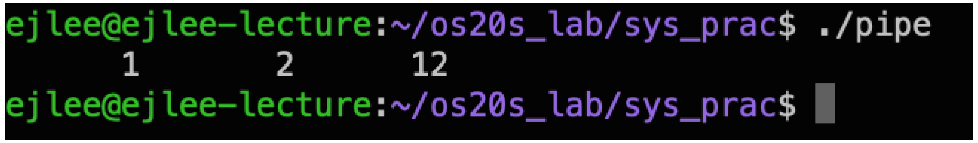


```
ejlee@ejlee-lecture:~/os20s_lab/sys_prac$ echo "hello world" | wc
      1       2      12
ejlee@ejlee-lecture:~/os20s_lab/sys_prac$ 
```



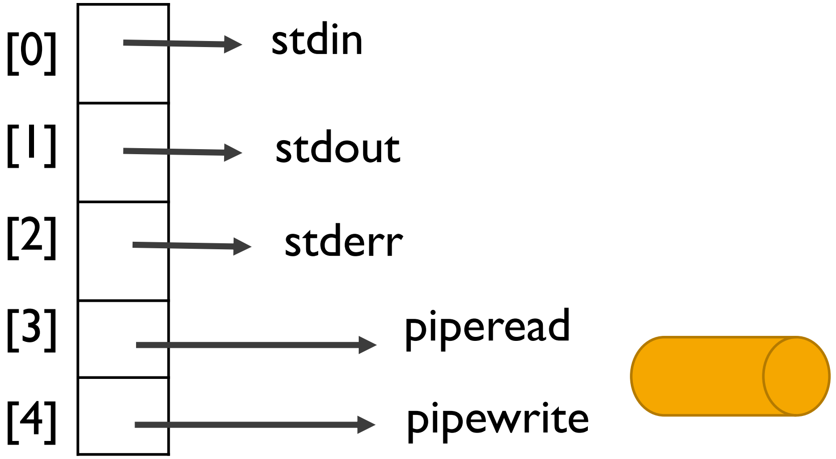
Pipes

```c
// pipe.c
int main()
{
    int p[2];
    char* argv[2];

    argv[0] = "wc";
    argv[1] = 0;

    pipe(p); // p[0]: readfd, p[1]: writefd
    if(fork() == 0) {
        close(0);
        dup(p[0]);
        close(p[0]);
        close(p[1]); // close unused write end
        execve("/usr/bin/wc", argv, NULL);
    } else {
        close(p[0]); // close unused read end
        write(p[1], "hello world\n", 12);
        close(p[1]);
    }
}
```



```
ejlee@ejlee-lecture:~/os20s_lab/sys_prac$ ./pipe
      1       2      12
ejlee@ejlee-lecture:~/os20s_lab/sys_prac$
```

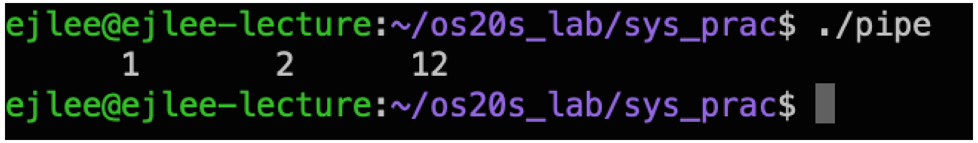
Before fork ..



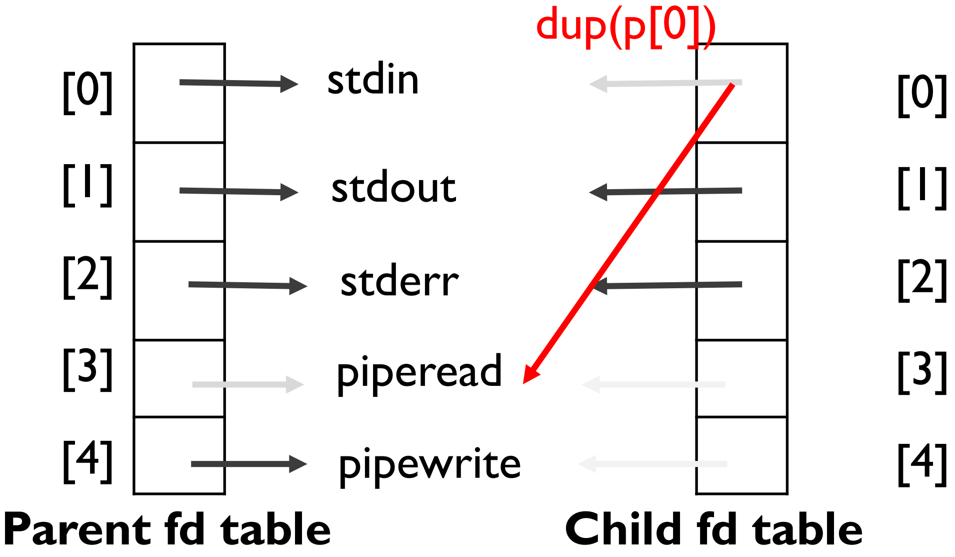
Parent fd table

Pipes

```c
// pipe.c
int main()
{
    int p[2];
    char* argv[2];

    argv[0] = "wc";
    argv[1] = 0;

    pipe(p); // p[0]: readfd, p[1]: writefd
    if(fork() == 0) {
        close(0);
        dup(p[0]);
        close(p[0]);
        close(p[1]); // close unused write end
        execve("/usr/bin/wc", argv, NULL);
    } else {
        close(p[0]); // close unused read end
        write(p[1], "hello world\n", 12);
        close(p[1]);
    }
}
```



```
ejlee@ejlee-lecture:~/os20s_lab/sys_prac$ ./pipe
      1       2      12
ejlee@ejlee-lecture:~/os20s_lab/sys_prac$ 
```



dup(p[0])
[0]
[1]
[2]
[3]
[4]
stdin
stdout
stderr
piperead
pipewrite
[0]
[1]
[2]
[3]
[4]
Parent fd table
Child fd table