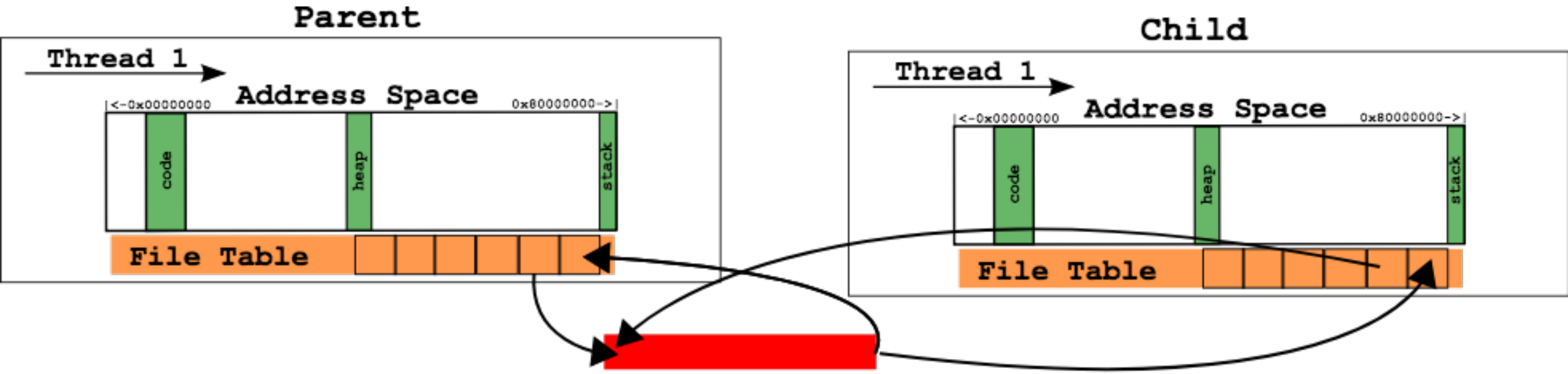
CS 1217
Operating Systems

Lecture 6 – Process Interface wrap up, CPU multiplexing; Interrupt handling

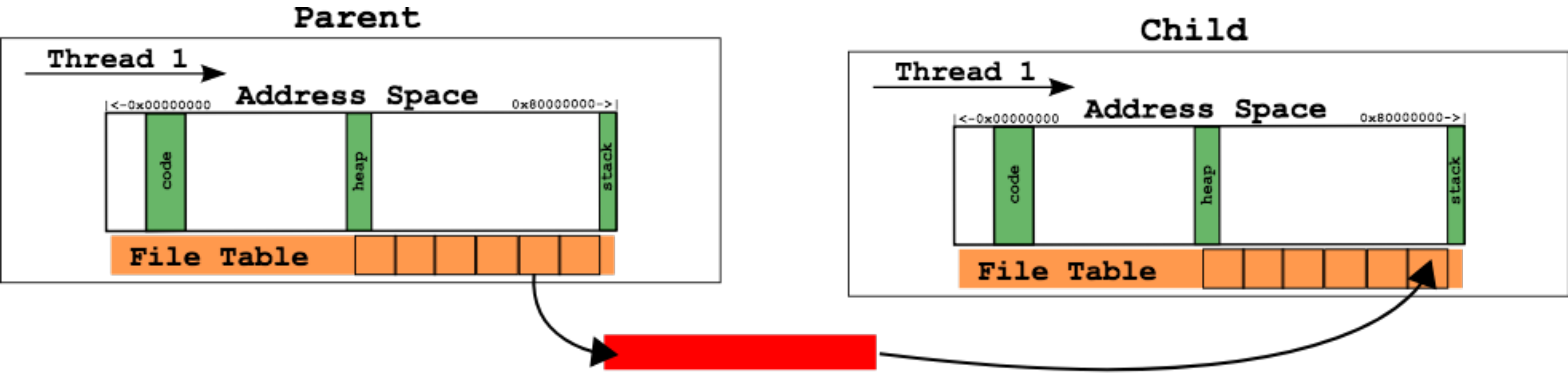
Logistics

- Assignment 2 is due tonight, please submit **before** time
- Lecture Rooms
    - Tuesday: AC01-207 (LT)
    - Wednesday: AC02-011 (LH)
- Lab Hours : Tuesday 6:30 pm – 8pm
    - Mandatory attendance
    - **AC03-005 (LT) [100]**
- Assignment 3 is coming out tomorrow
    - Make sure to show up to lab hours today.
    - Not difficult, but will require a lot of reading + trial and error

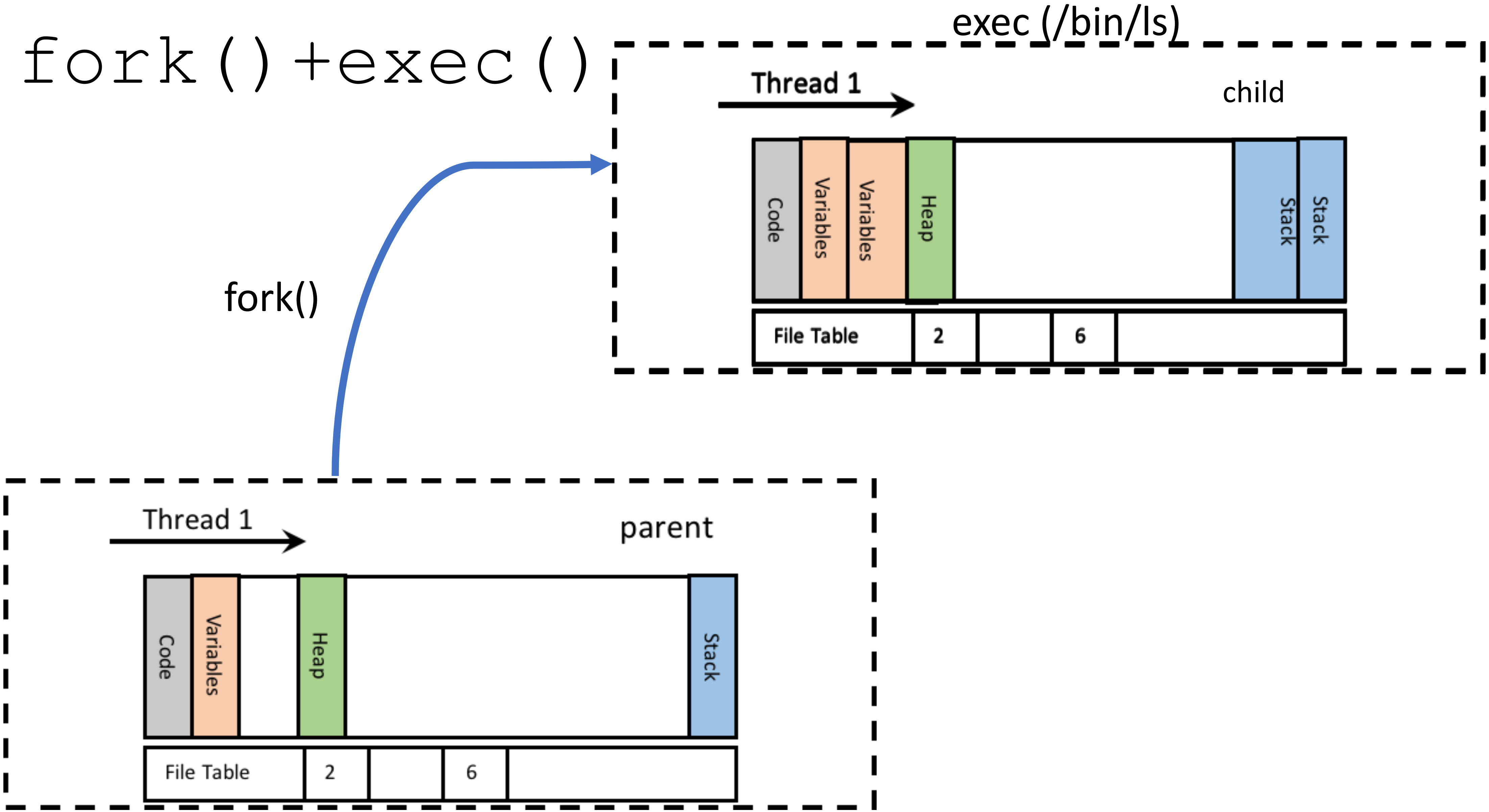
IPC using `fork()` and `pipe()`



IPC using `fork()` and `pipe()`

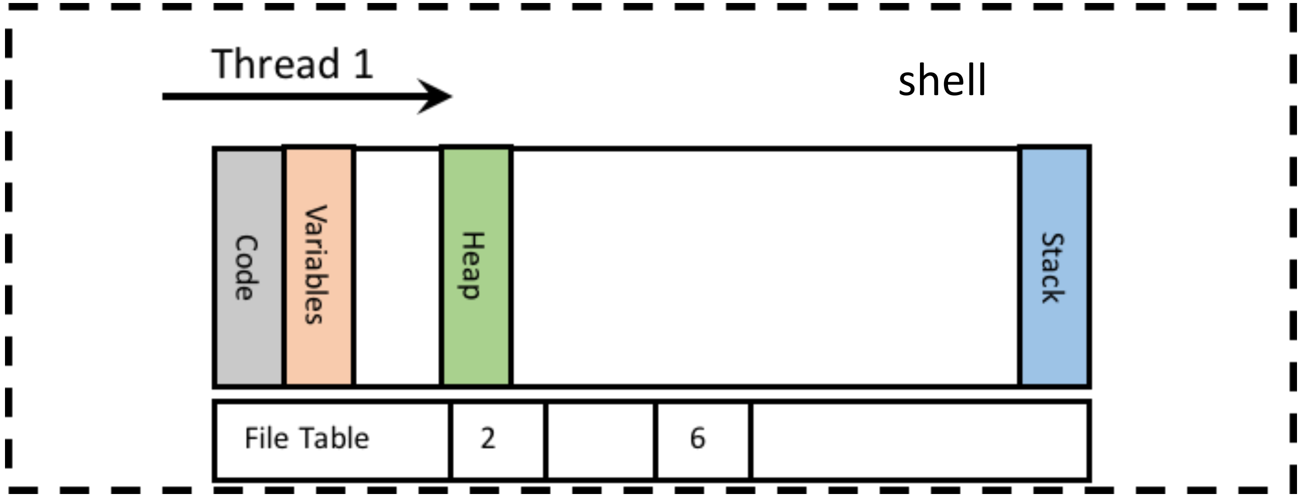


fork()+exec()


exec (/bin/ls)
Thread 1
child
Code
Variables
Variables
Heap
Stack
Stack
File Table
2
6
fork()
Thread 1
parent
Code
Variables
Heap
Stack
File Table
2
6

What happens here?

exec (/bin/ls)



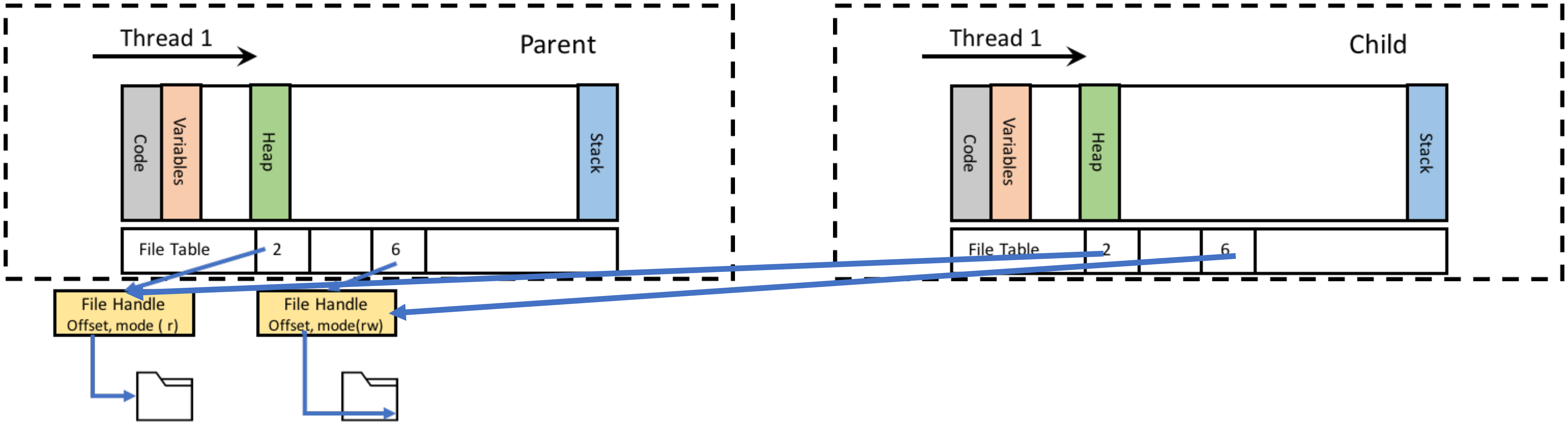
More `exec()` details

- What happens if a process’s parent exits before it does?

- The "orphaned" process is assigned the `init` process as a parent, which will collect its exit code when it exits. Referred to as *reparenting*.

More `exec()` details

- `exec()` might not want to modify the File Table of the process. Why?
- Can reuse the work done by `fork()` for duplicating file handles



Process End of Life

- Can call `exit()` and exit gracefully, at the time of its choosing
- The exiting process can pass an `exit()` code
- This code sits with the kernel to be retrieved by the parent process

- What can this code be used for?

- The parent can use this code to take some action : `wait()`
  - How does this work in the case of shell?

More on exit()

- A parent process receives a SIGCHILD signal if a child calls exit().
  - It can retrieve the child’s exit code if it so wishes

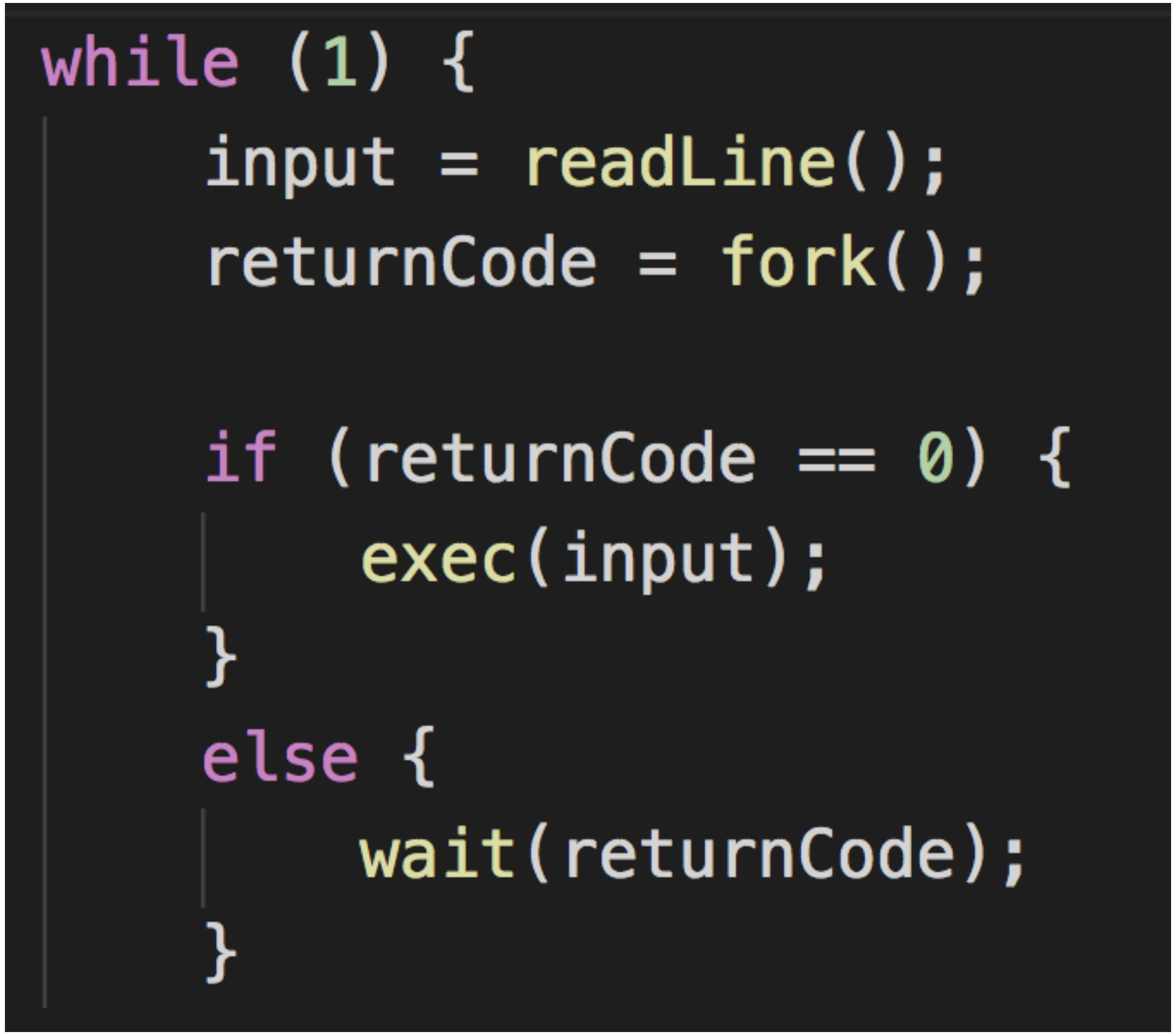
- Try `disown` on bash
  - Allows children to continue running even if bash exits

`wait()` types

- **Blocking `wait()`**
  - block until the child exits, unless it has already exited in which case it returns immediately.

- **Non-Blocking `wait()`**
  - do not block. Instead, its return status indicates if the child has exited and, if so, what the exit code was

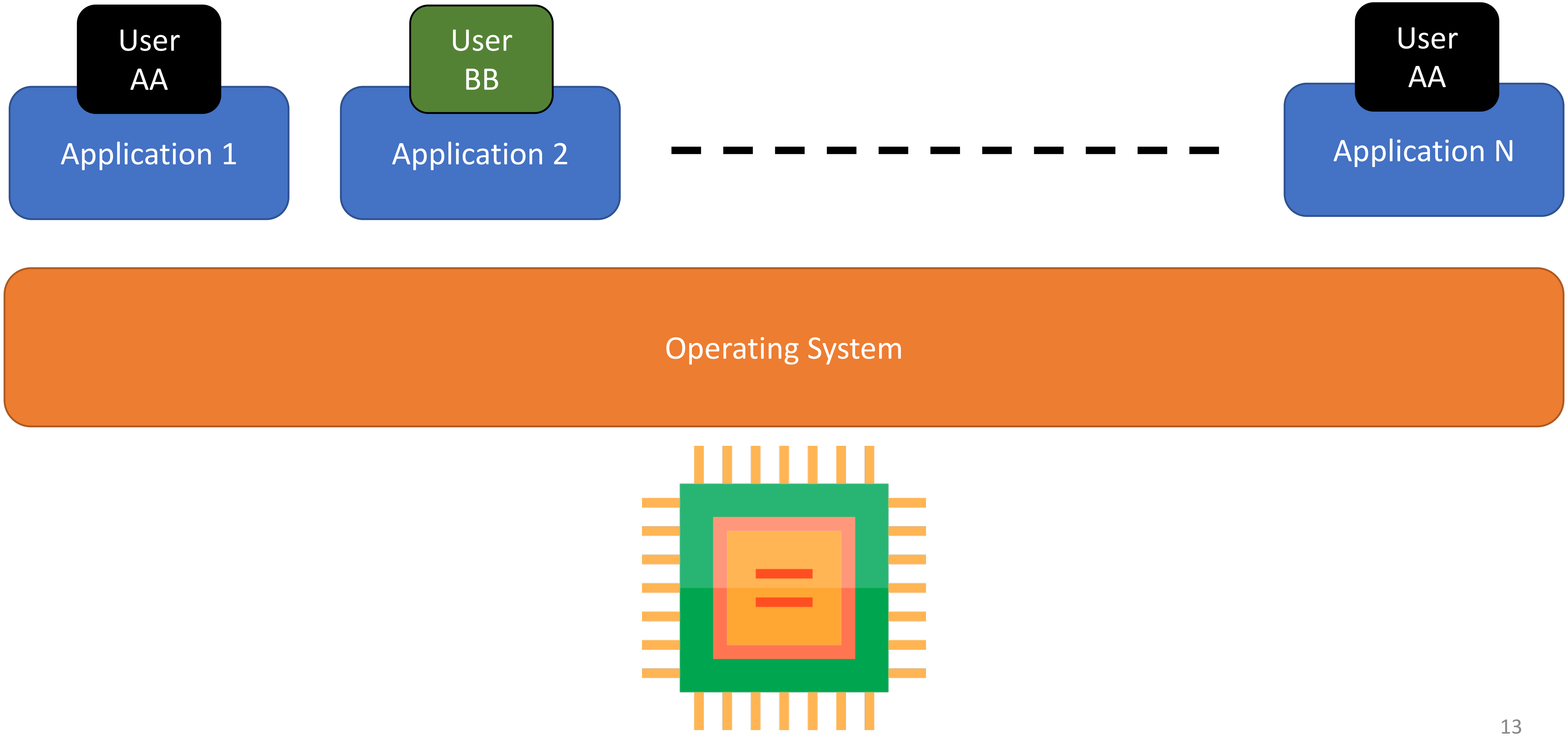
Simple shell



```
while (1) {
    input = readLine();
    returnCode = fork();

    if (returnCode == 0) {
        exec(input);
    }
    else {
        wait(returnCode);
    }
```

Virtualizing the CPU



User AA

Application 1

User BB

Application 2

User AA

Application N

Operating System

Virtualizing the CPU

- CPU is a resource that is used by all processes
  - To keep discussion simple, we will assume a single core for now
- The processes could be from multiple users
  - Need to give each user a fair share of the resource
- This resource needs to be shared across multiple processes that are running “concurrently”
- The processes might want the kernel to carry out ***privileged*** operations on their behalf
  - Reading from disk, getting packet from network etc.