CSCI315 – Operating Systems Design

Department of Computer Science
Bucknell University

Processes: Concepts and their Creation

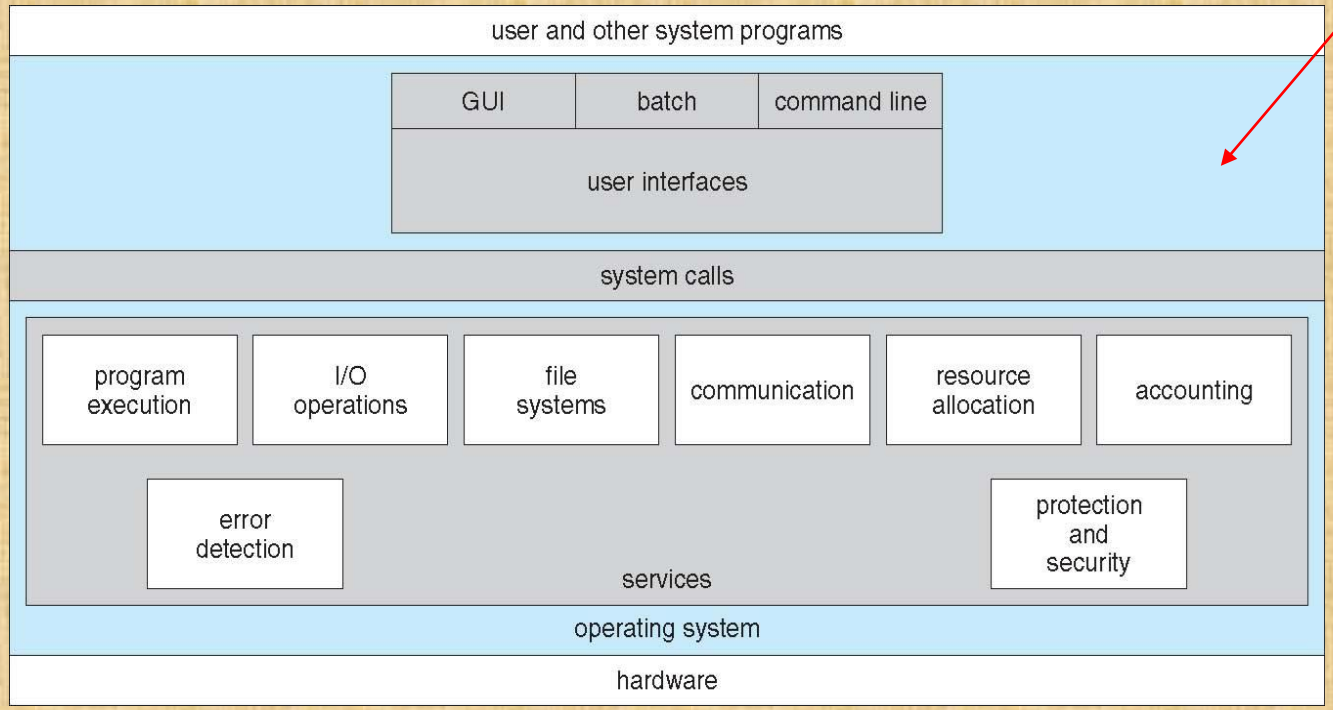
Ch 3

This set of notes is based on notes from the textbook authors, as well as L. Felipe Perrone, Joshua Stough, and other instructors.
Xiannong Meng, Fall 2021.

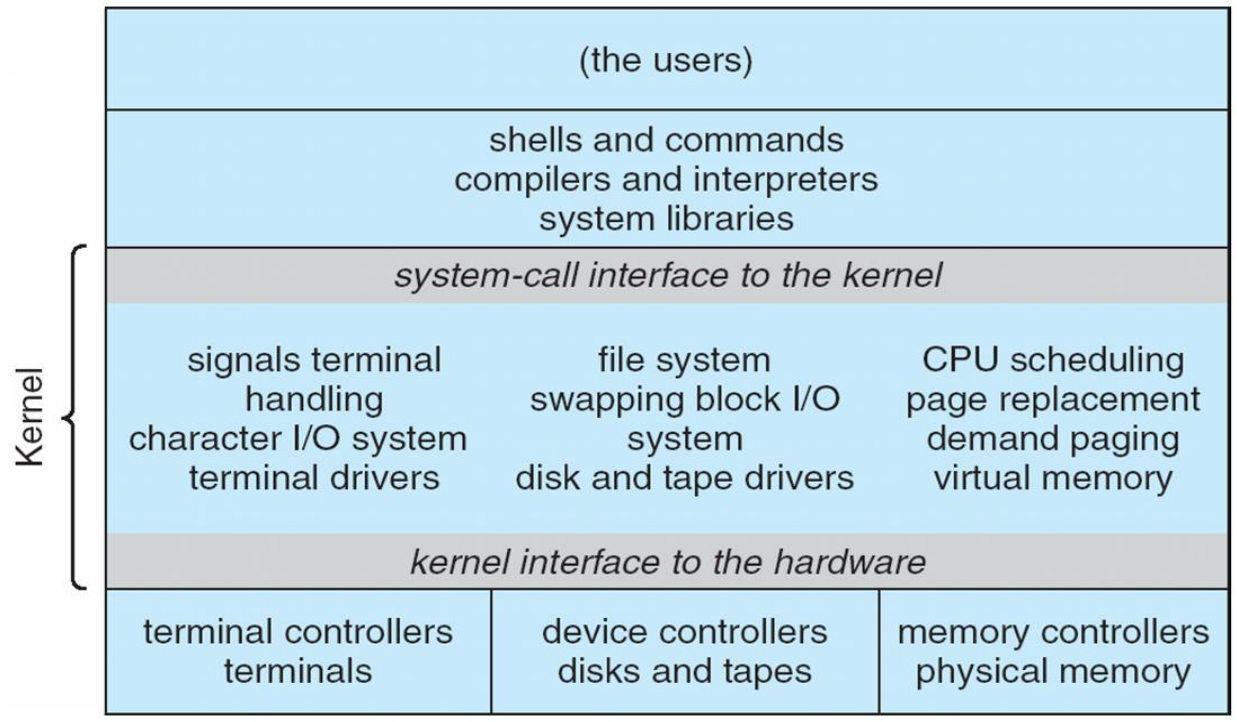
We discuss the basic concept and creation of a process first, other topics later to fit the lab schedule.

OS Services

The entire blue area is operating system.

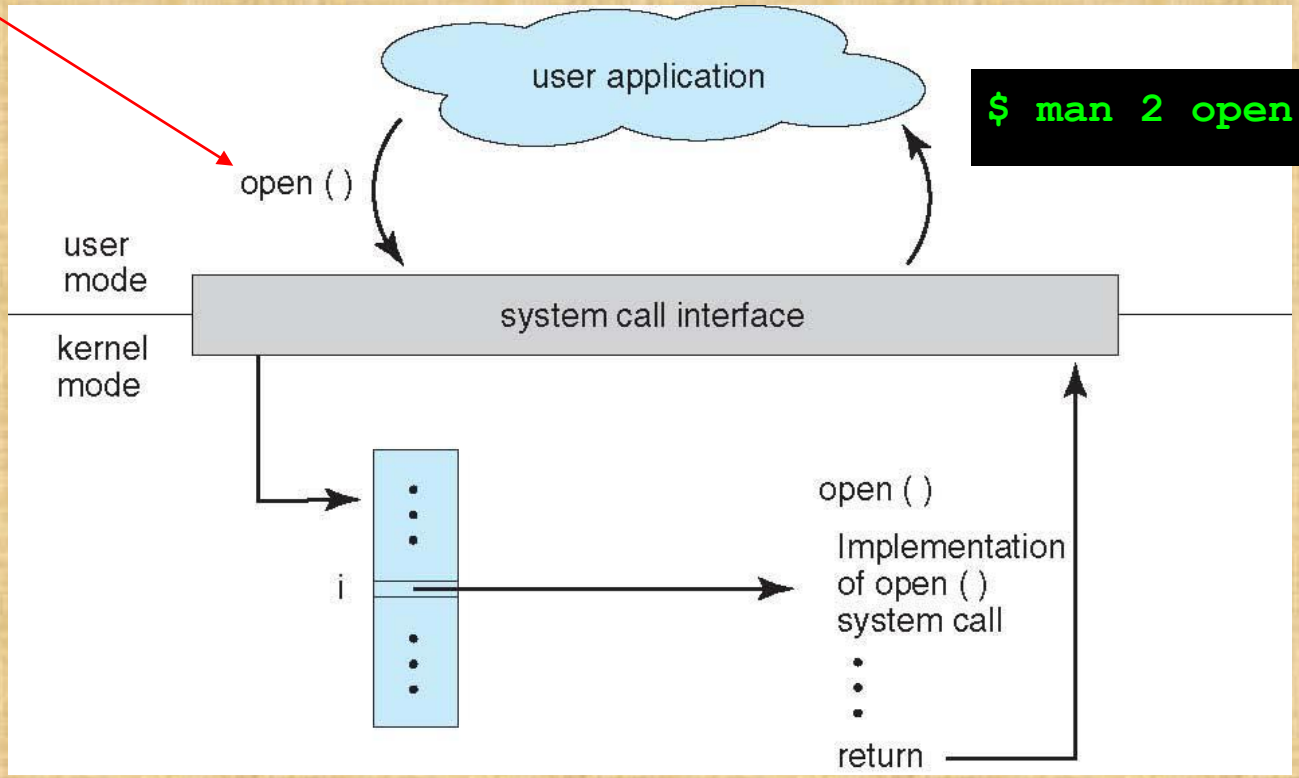


Unix Structure



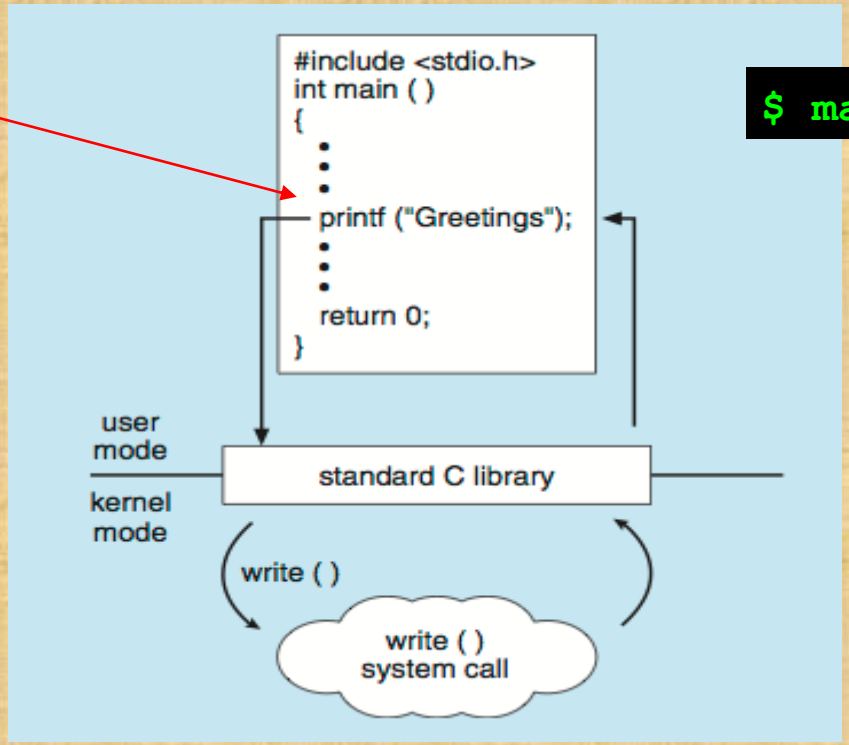
System Calls and the OS

Take the “open()” call as an example.



System Calls and Libraries

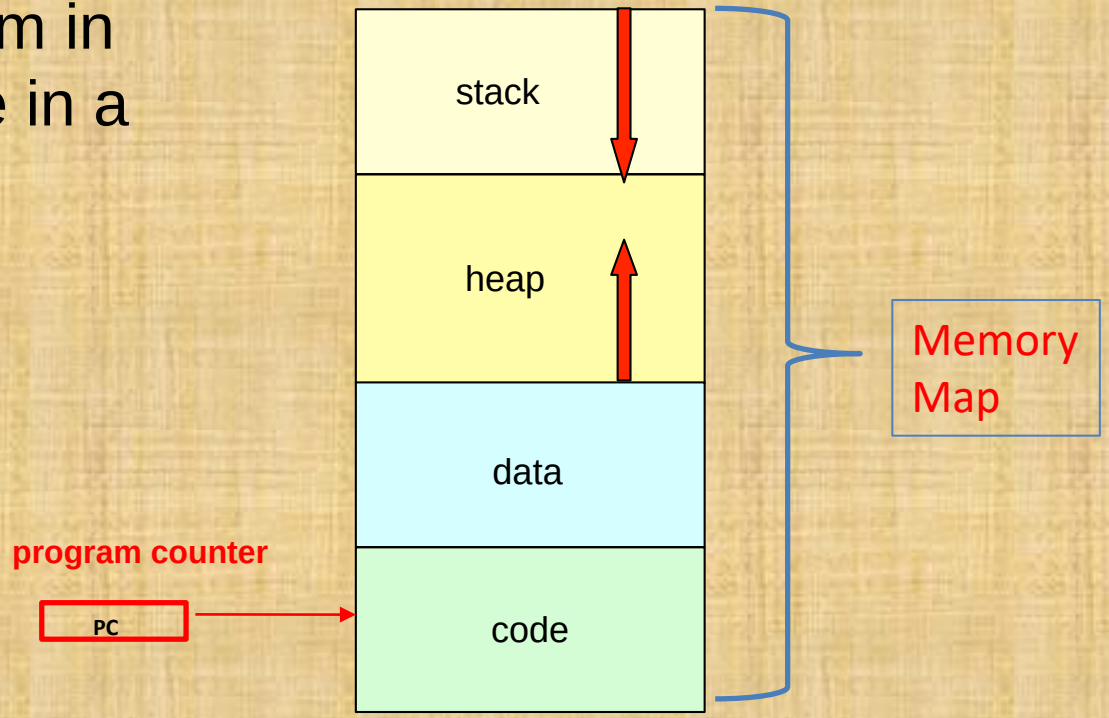
"printf()" is a C library function.



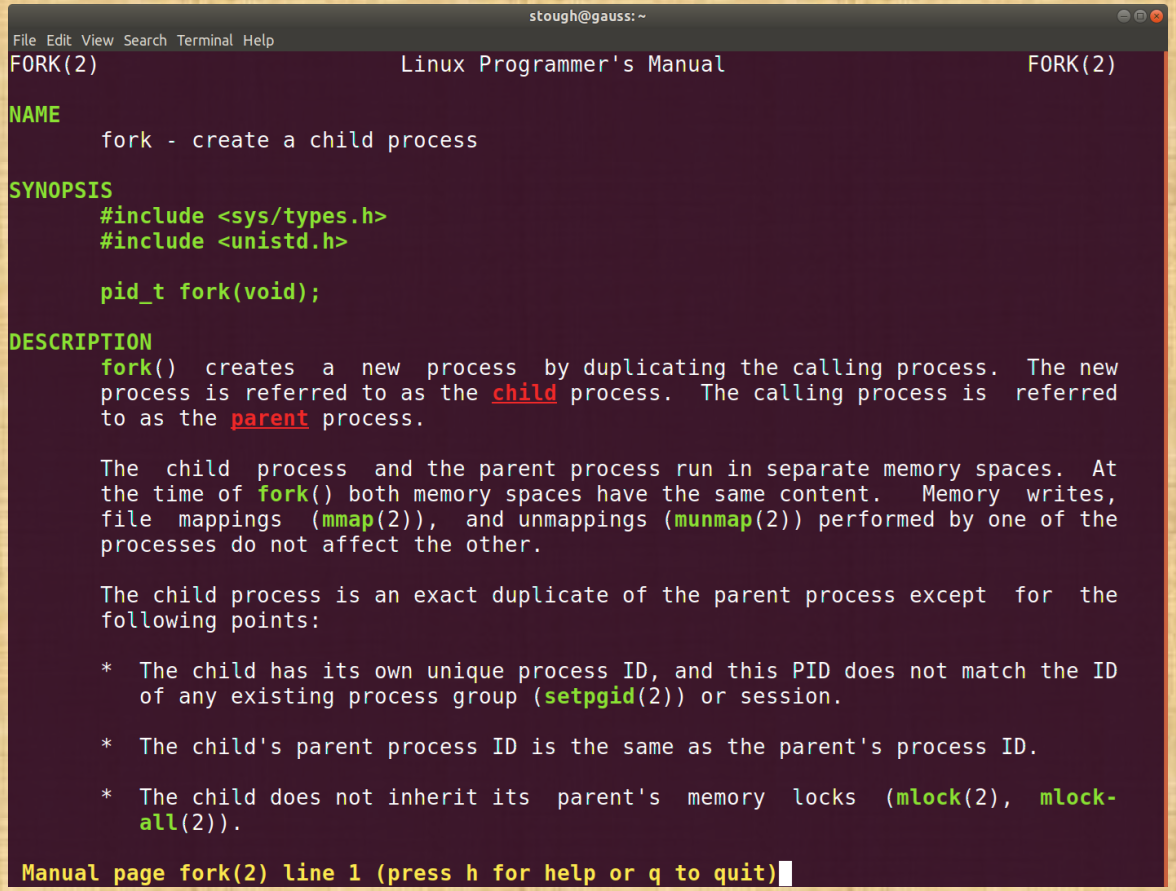

```
$ man 3 printf
```

Process Concept

- Process – a program in execution; the code in a process executes sequentially.
- A process includes:
  - program counter,
  - code,
  - stack,
  - heap,
  - data section.

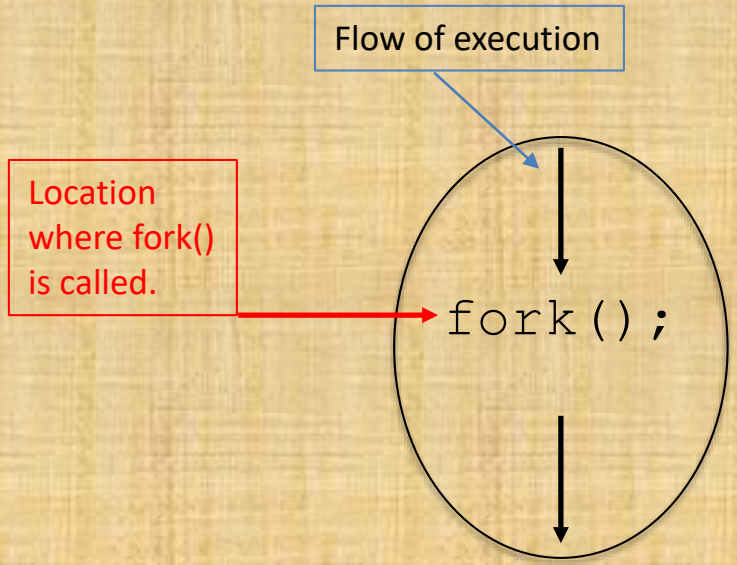


All *processes* in Unix are created through `fork()`



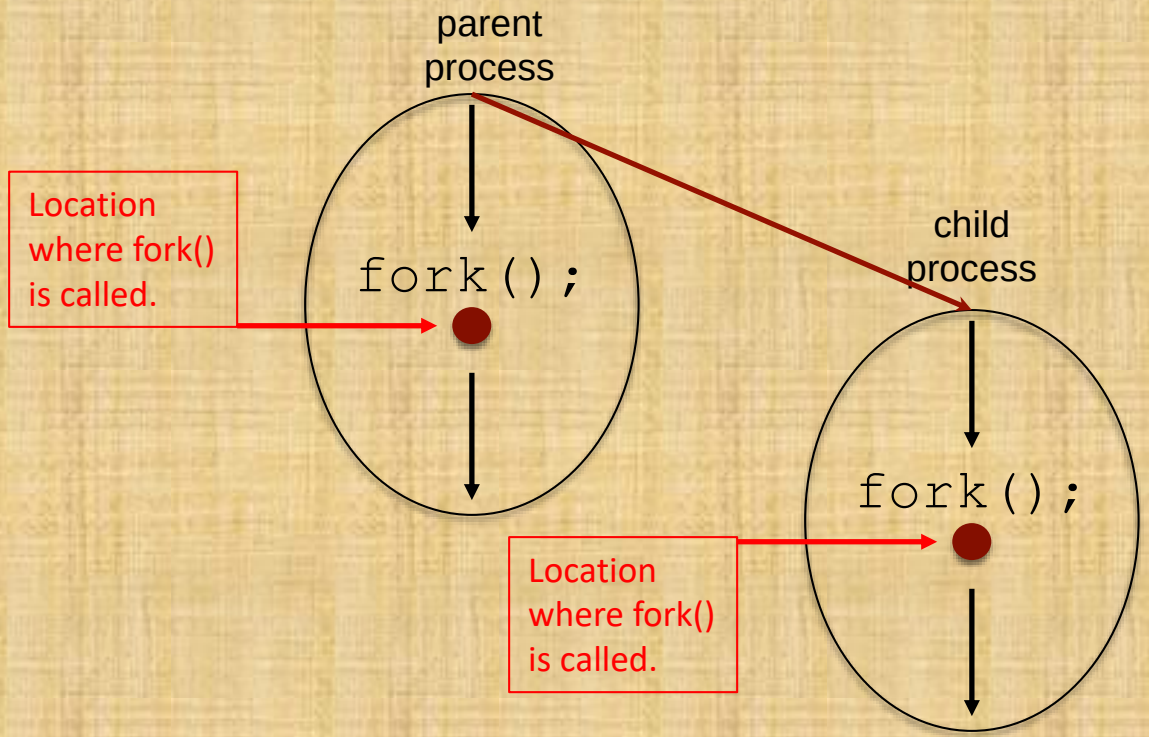
Forking

(yeah, it's a thing, a Unix thing)



Forking

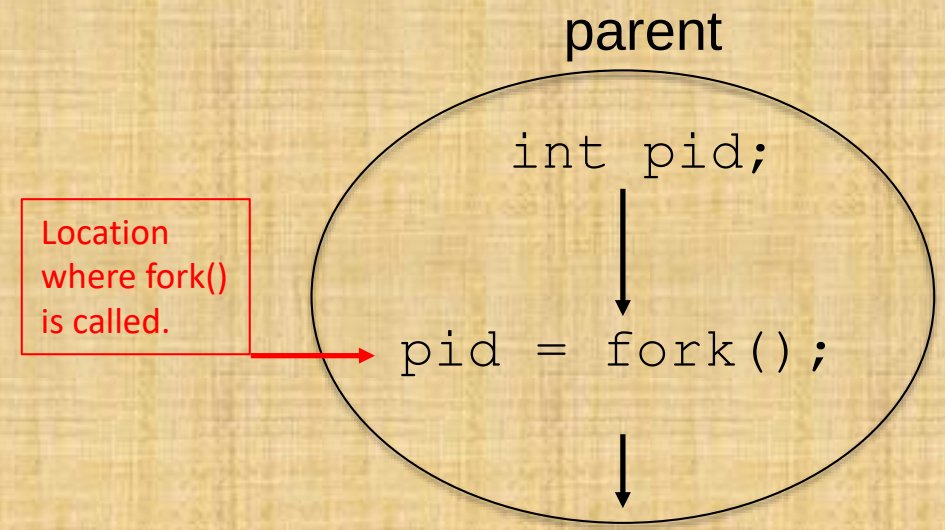
(yeah, it's a thing, a Unix thing)



Note that the child process and the parent process are **identical** at the moment of calling fork(). They differ when execution starts.

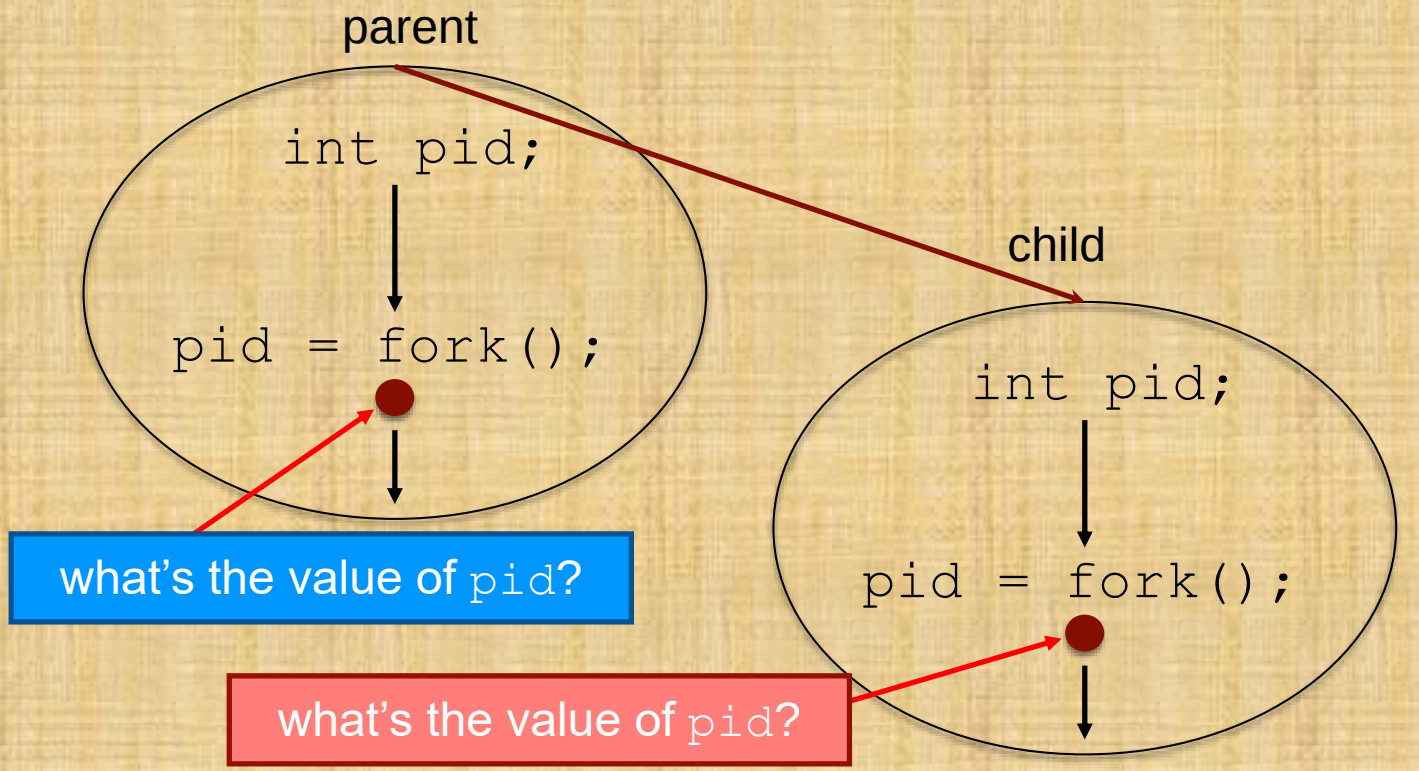
Forking

(what that return value is for)



Forking

(what that return value is for)

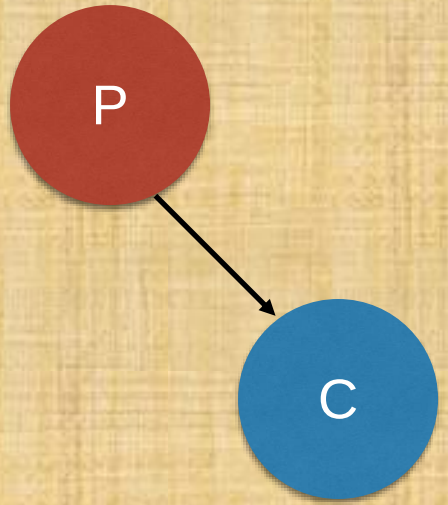


Using `fork` safely

```
int pid;
…
pid = fork();
if (0 != pid) {
   // code of the parent
   …
} else {
   // code of the child
   …
}
…
```

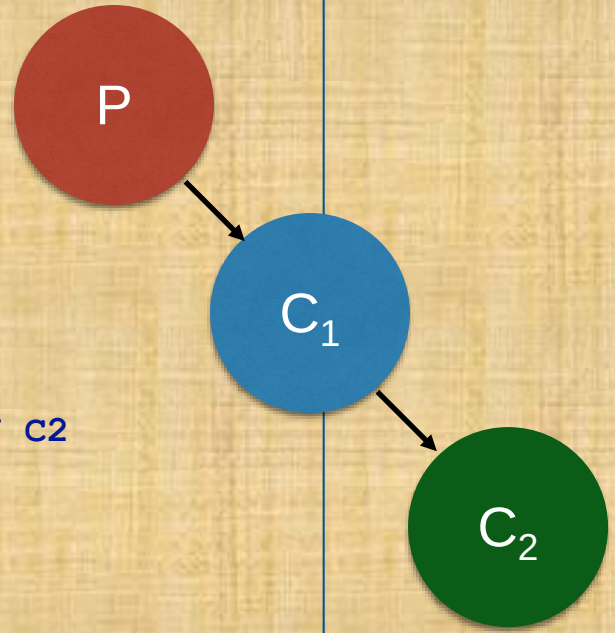
Using `fork` safely

```
int pid;
…
pid = fork();
if (0 != pid) {
   // code of parent P
   …
} else {
   // code of child C
   …
}
…
```



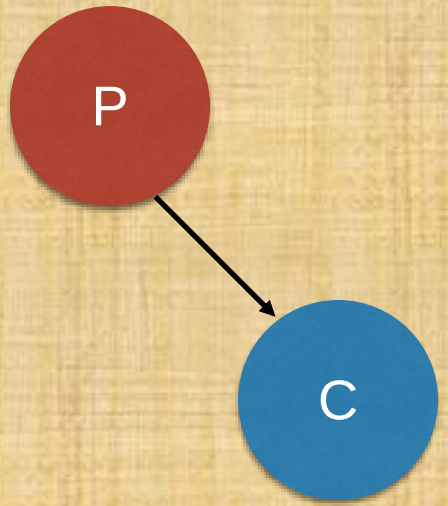
Using `fork` safely

```
int pid1, pid2;
pid1 = fork();
if (0 != pid1) {
  // code of parent P
  …
} else {
  pid2 = fork();
  // code of child C1
  if (0 != pid2) {
    // more code of child C1, parent of C2
    …
  } else {
    // code of child C2
    …
  }
```

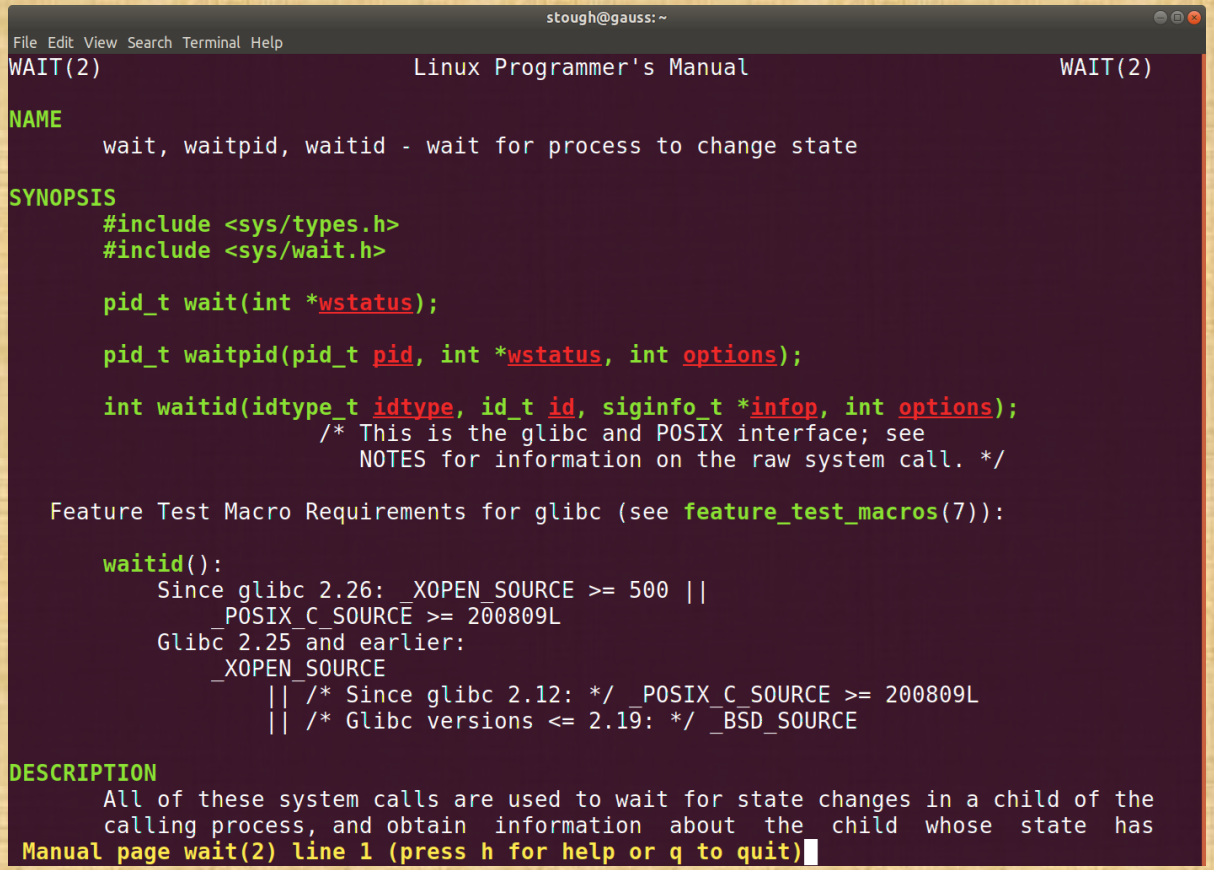


Using `fork` even more safely

```
int pid;
pid = fork();
if (-1 == pid) {
    // error handling
    …
} else if (0 != pid) {
   // code of parent P
   …
} else {
   // code of child C
   …
}
```

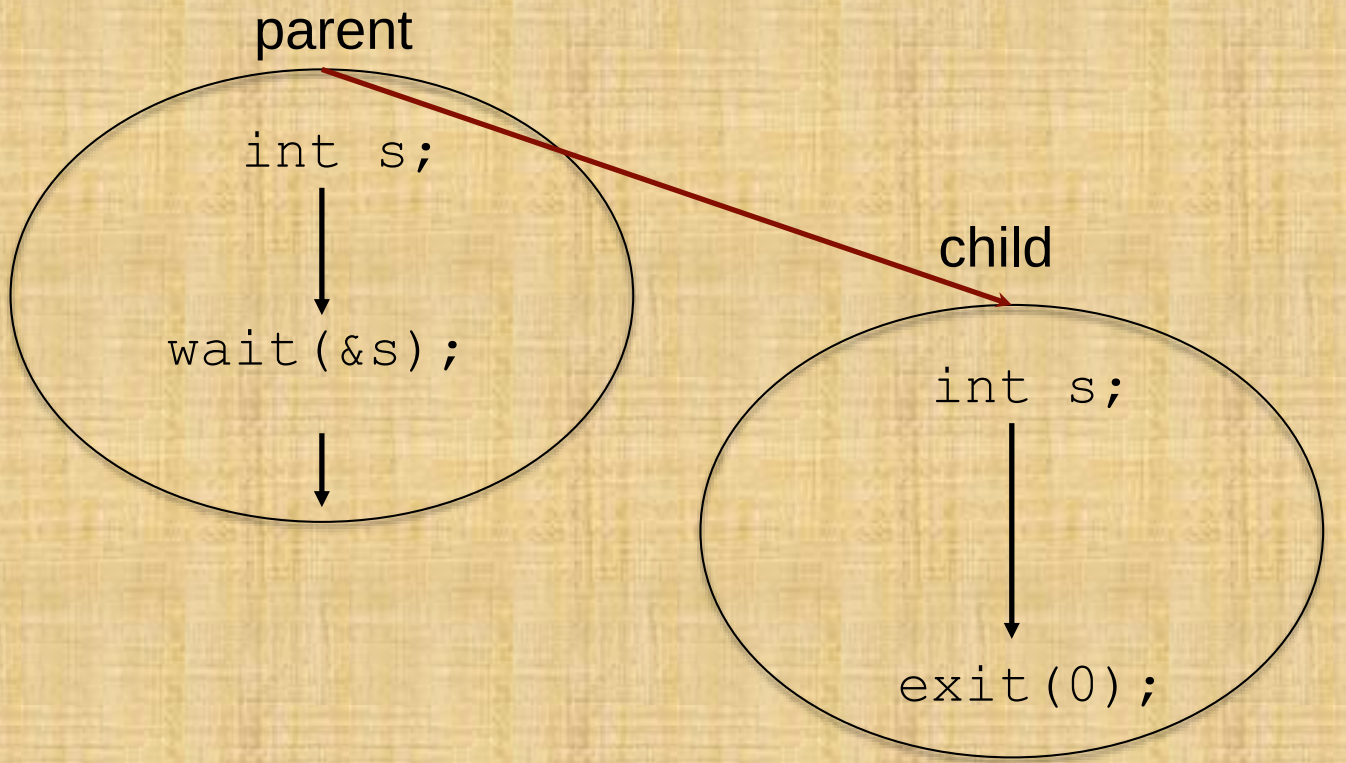


Joining *processes* in Unix



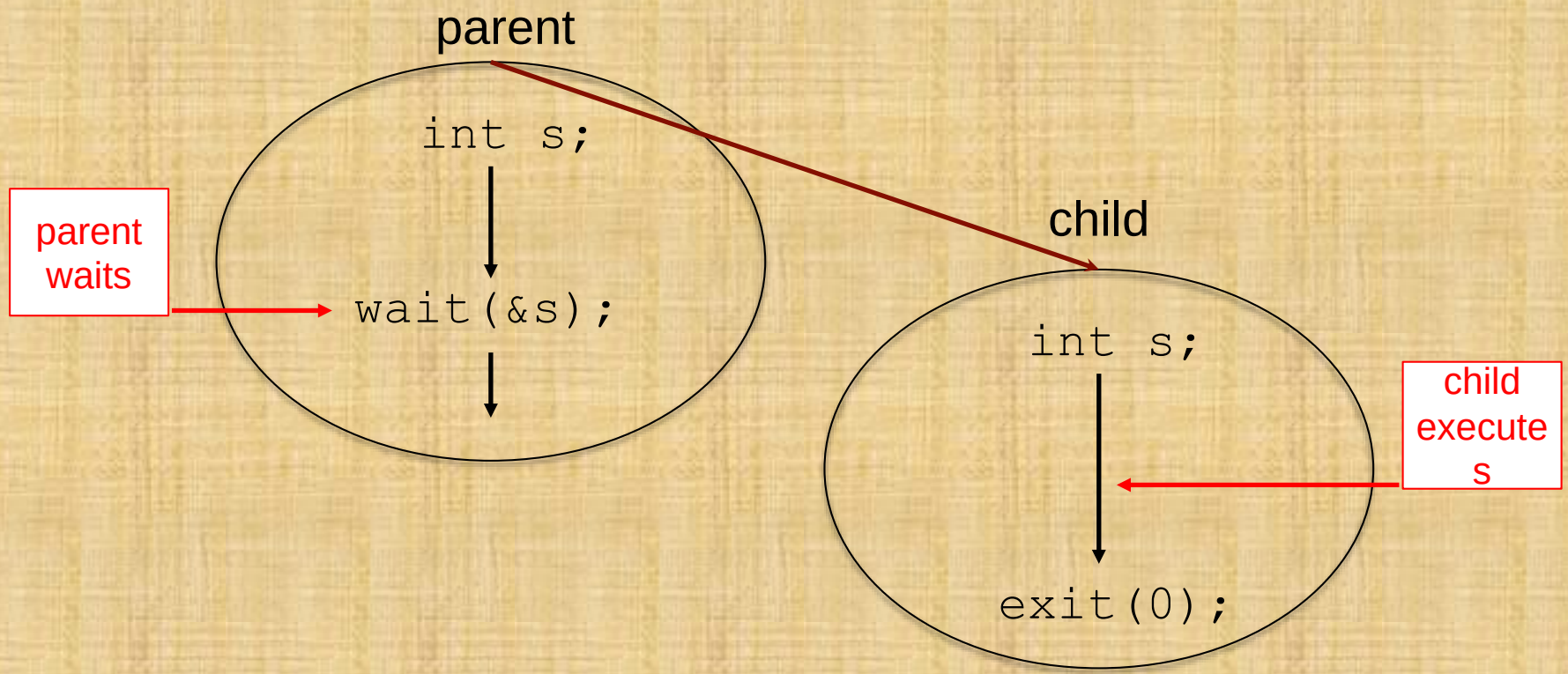
Waiting

(the inverse of forking)



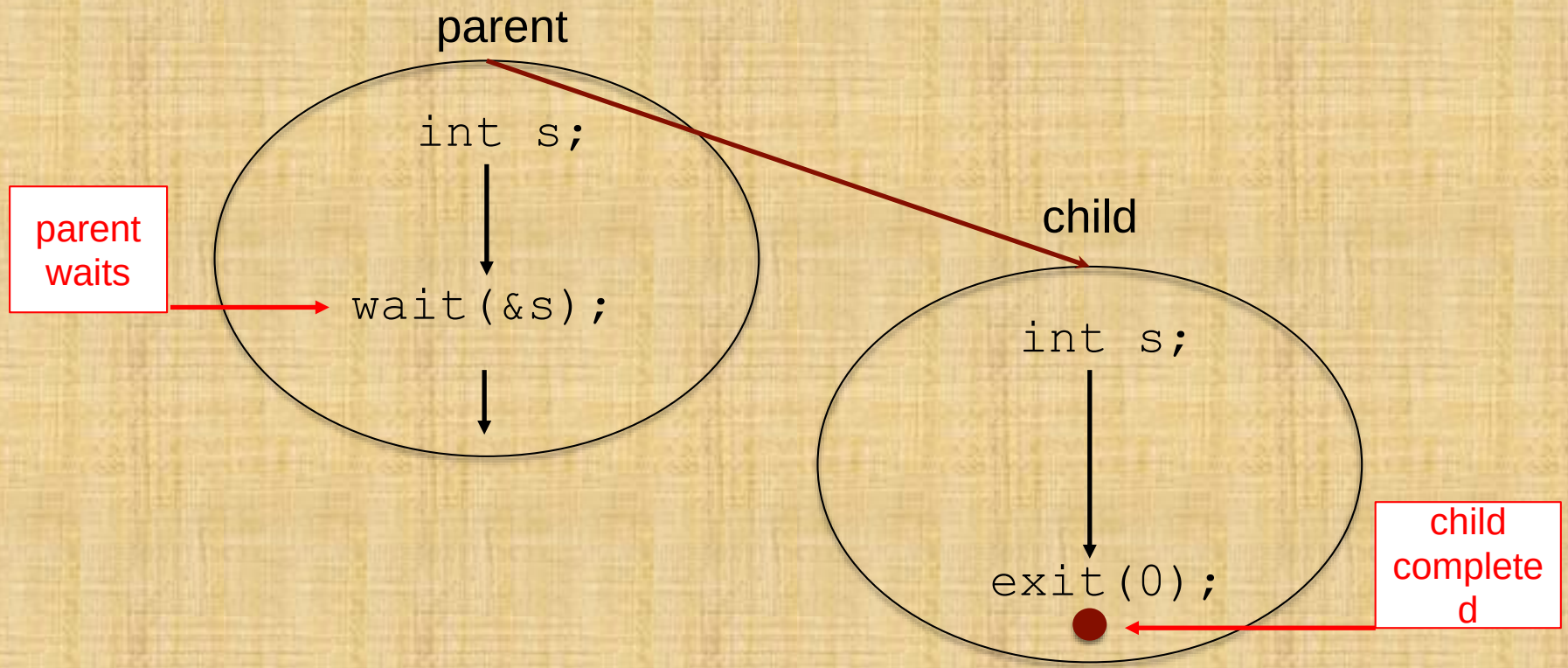
Waiting

(the inverse of forking)



Waiting

(the inverse of forking)



Waiting

(the inverse of forking)