

# CSCI315 – Operating Systems Design

Department of Computer Science  
Bucknell University

## Processes: Concepts and their Creation

Ch 3

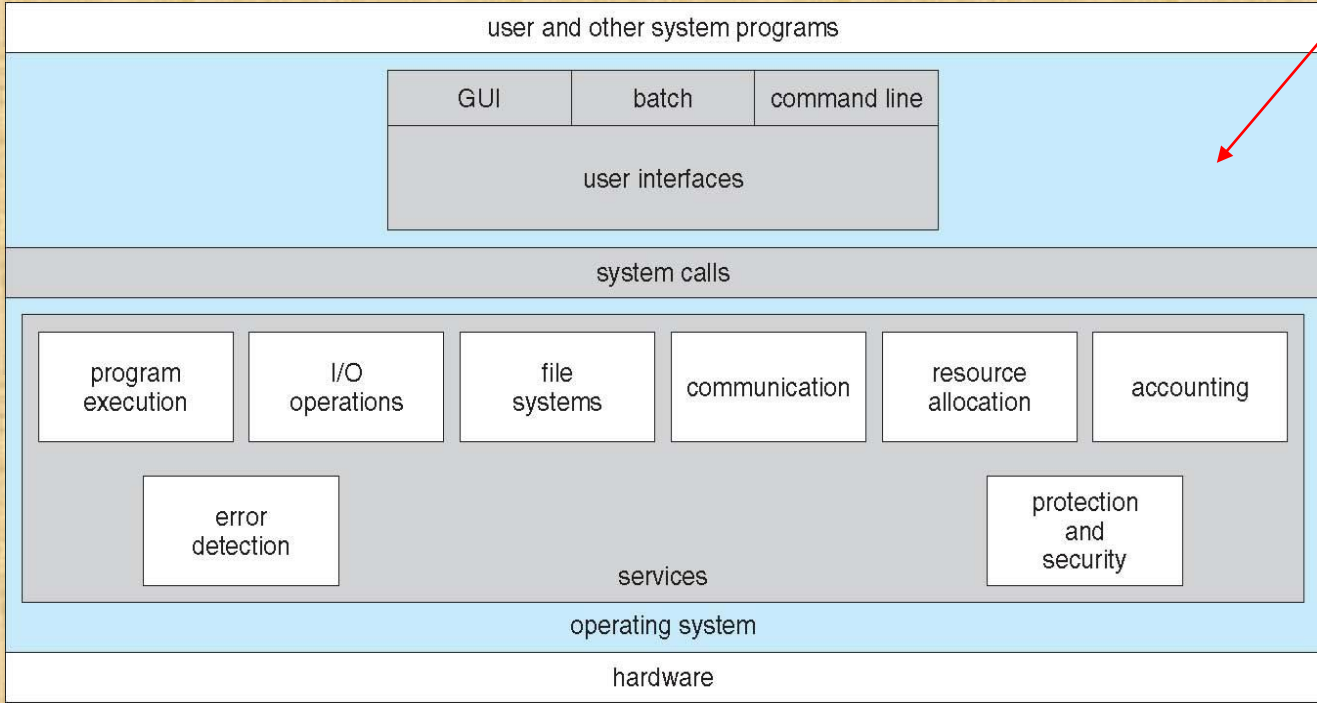
*This set of notes is based on notes from the textbook authors, as well as L. Felipe Perrone, Joshua Stough, and other instructors.*

*Xiannong Meng, Fall 2021.*

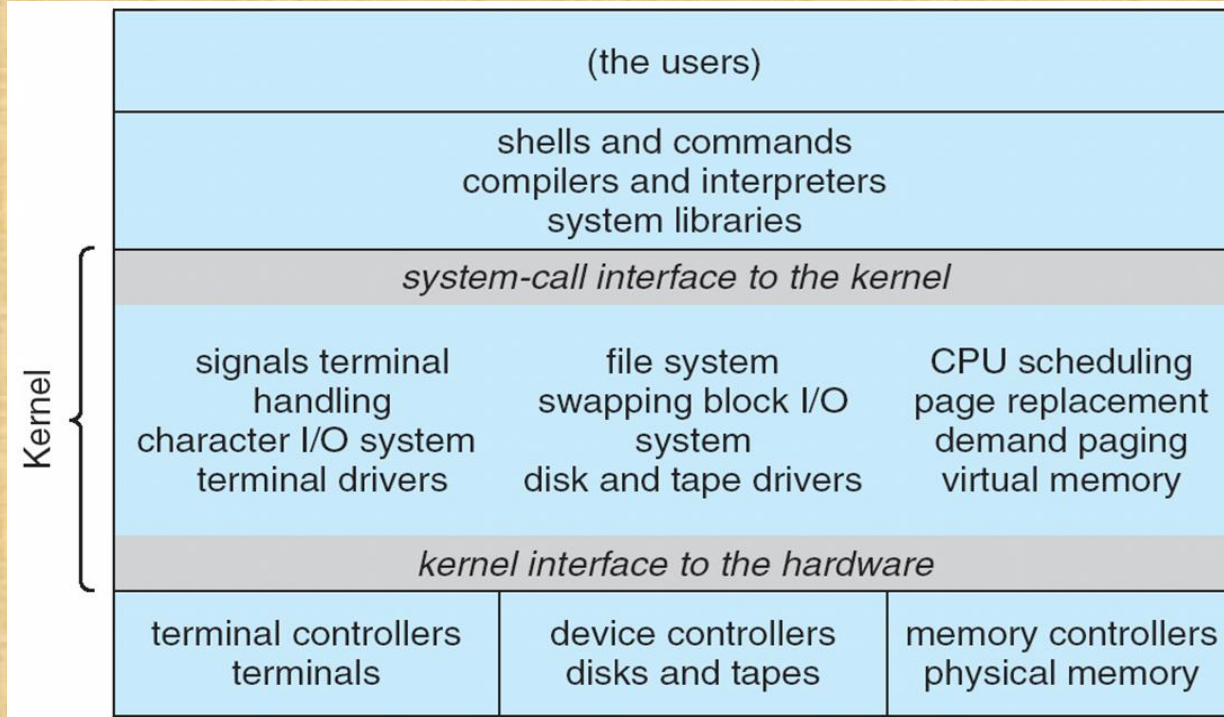
**We discuss the basic concept and creation of a process first, other topics later to fit the lab schedule.**

# OS Services

The entire blue area is operating system.

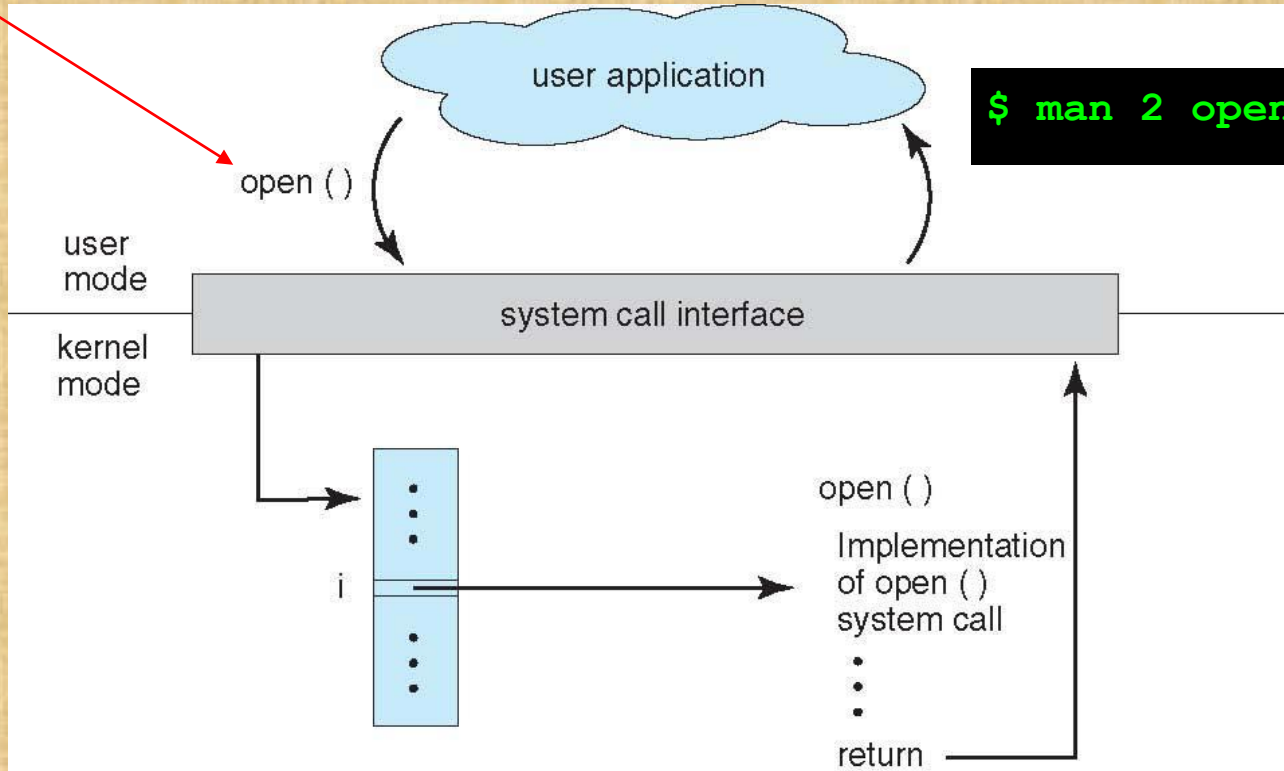


# Unix Structure



# System Calls and the OS

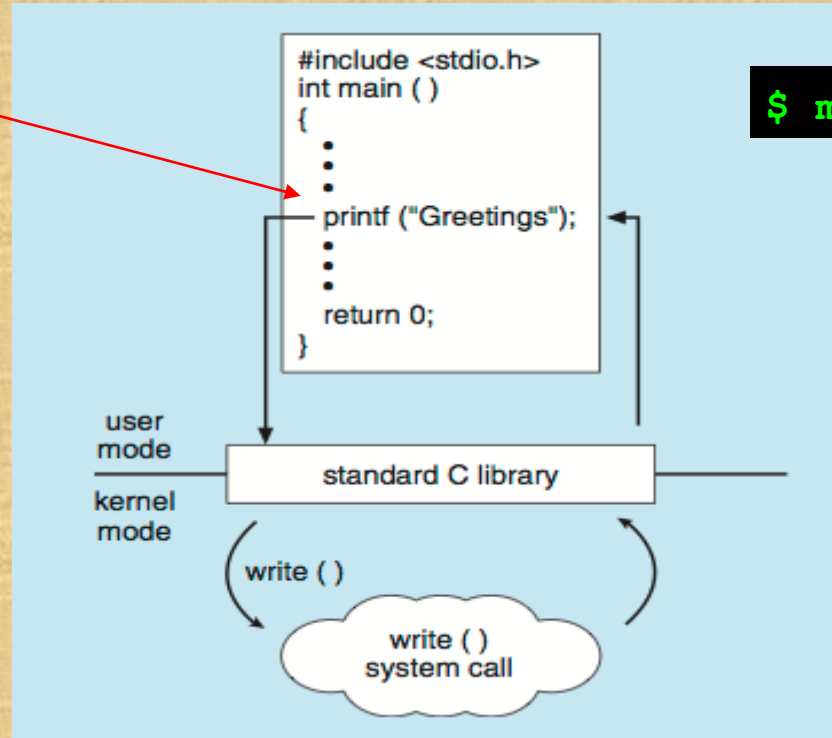
Take the "open()" call as an example.



# System Calls and Libraries

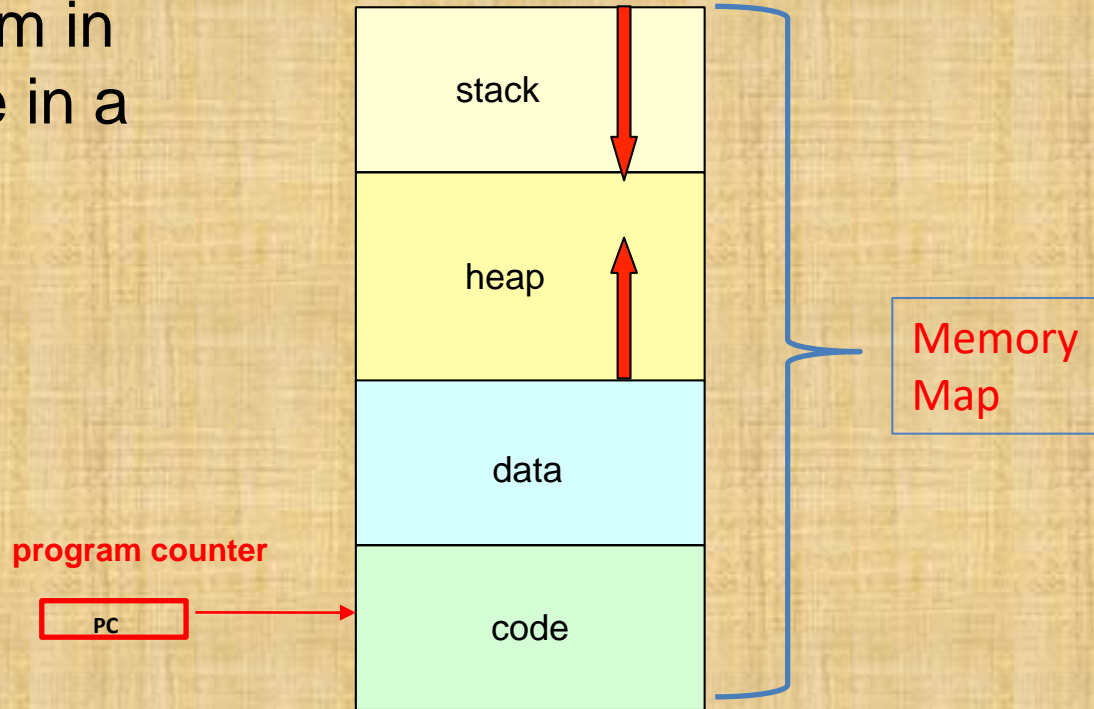
“printf()” is a C library function.

```
$ 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.



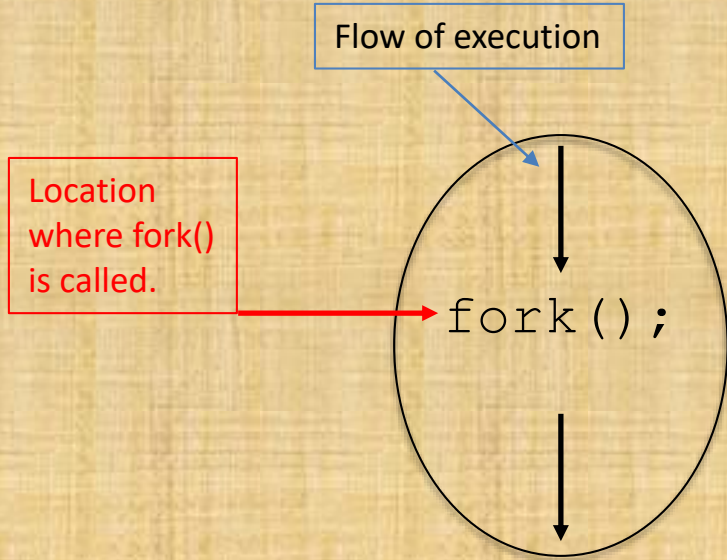
# All *processes* in Unix are created through `fork()`

```
stough@gauss:~  
File Edit View Search Terminal Help  
FORK(2) Linux Programmer's Manual FORK(2)  
  
NAME  
fork - create a child process  
  
SYNOPSIS  
#include <sys/types.h>  
#include <unistd.h>  
  
pid_t fork(void);  
  
DESCRIPTION  
fork() creates a new process by duplicating the calling process. The new  
process is referred to as the child process. The calling process is referred  
to as the parent process.  
  
The child process and the parent process run in separate memory spaces. At  
the time of fork() both memory spaces have the same content. Memory writes,  
file mappings (mmap(2)), and unmappings (munmap(2)) performed by one of the  
processes do not affect the other.  
  
The child process is an exact duplicate of the parent process except for the  
following points:  
  
* The child has its own unique process ID, and this PID does not match the ID  
of any existing process group (setpgid(2)) or session.  
  
* The child's parent process ID is the same as the parent's process ID.  
  
* The child does not inherit its parent's memory locks (mlock(2), mlock-  
all(2)).  
  
Manual page fork(2) line 1 (press h for help or q to quit)
```



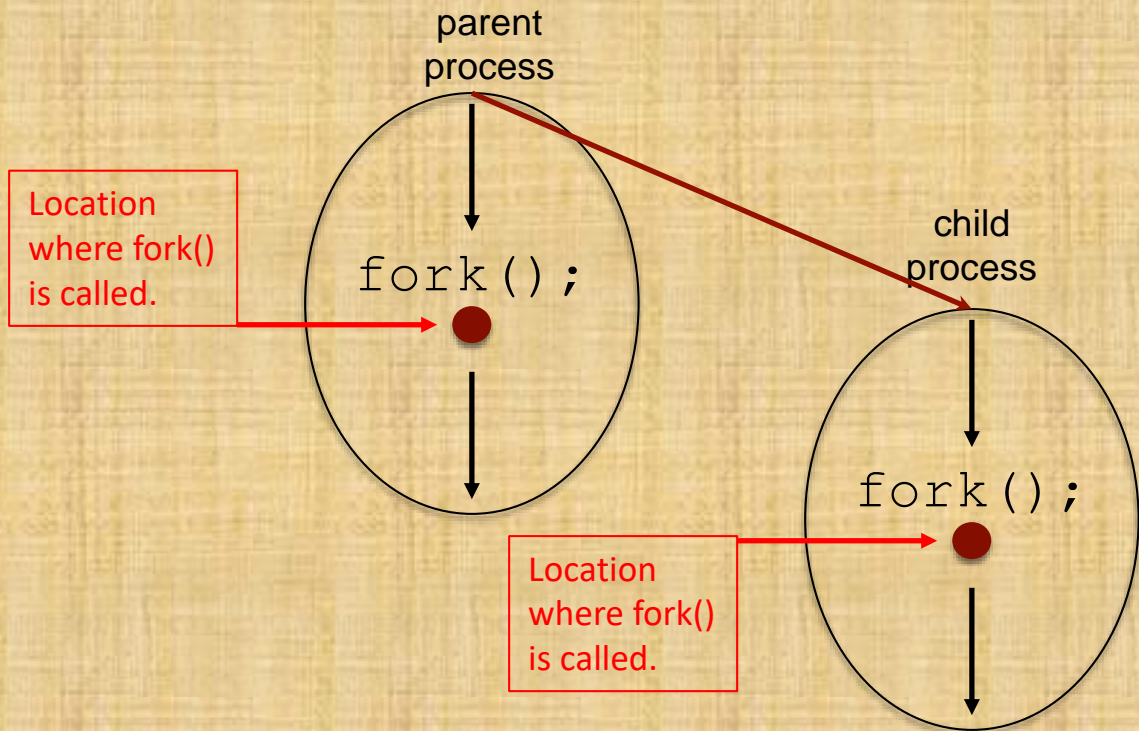
# Forking

(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)

parent

```
int pid;
```



```
pid = fork();
```

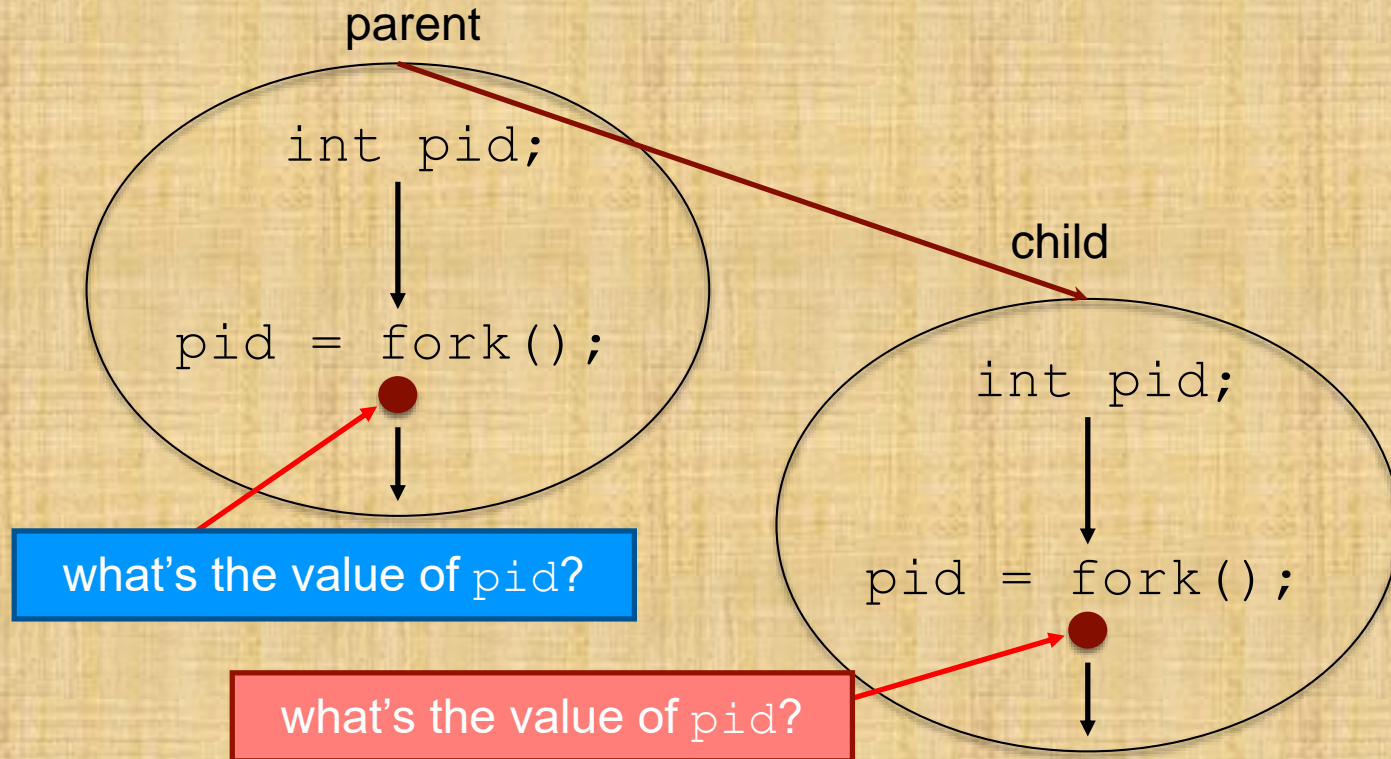


Location  
where fork()  
is called.



# Forking

(what that return value is for)

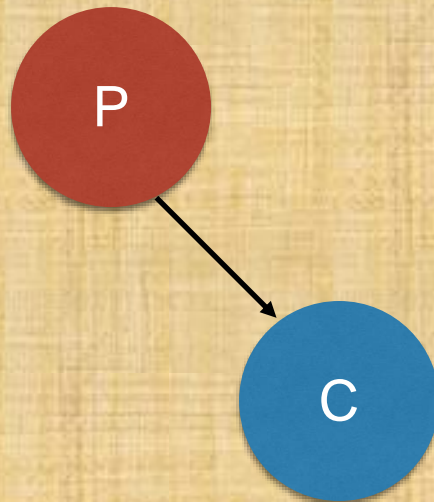


# Using `fork` safely

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

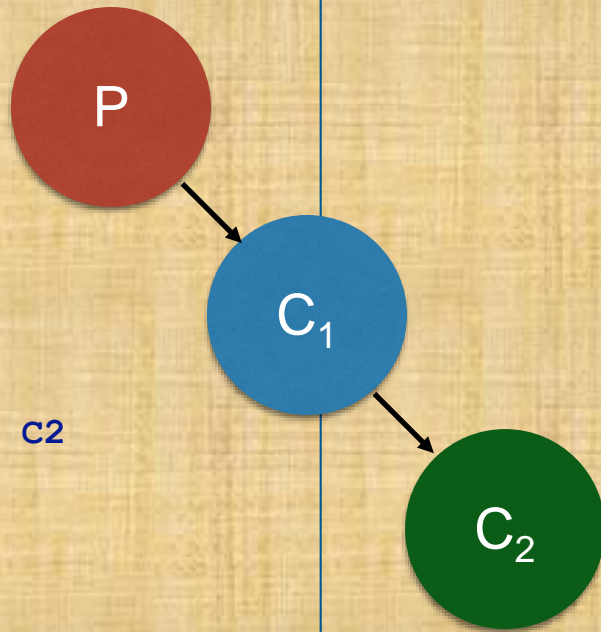
# Using `fork` safely

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



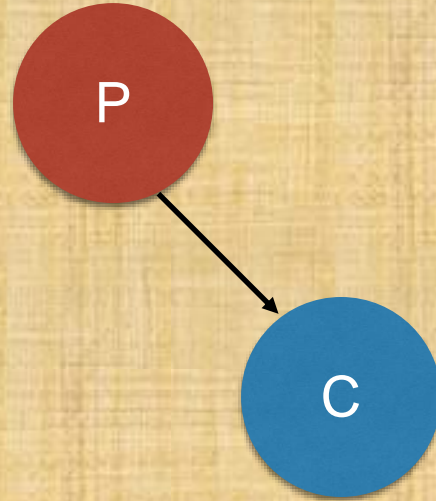
# Using fork safely

```
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
    ...
}
```



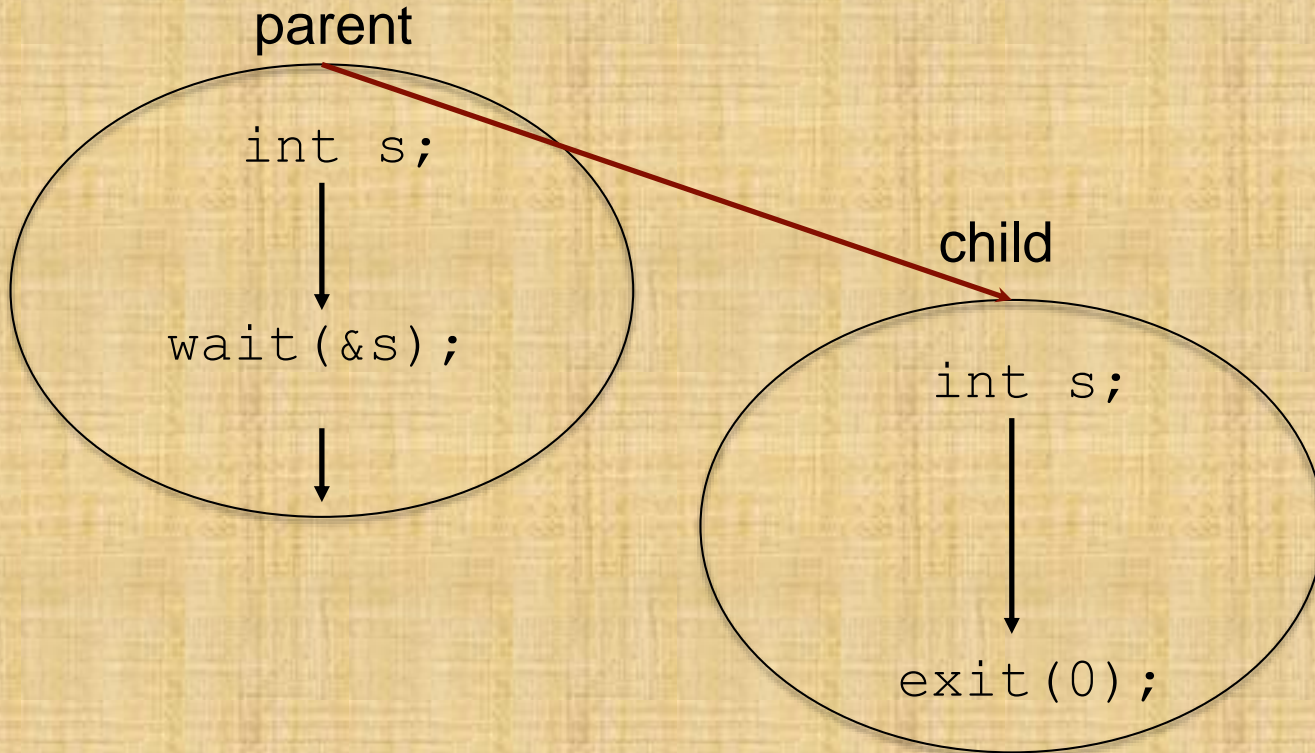


# Joining *processes* in Unix

```
stough@gauss:~  
File Edit View Search Terminal Help  
WAIT(2) Linux Programmer's Manual WAIT(2)  
  
NAME  
wait, waitpid, waitid - wait for process to change state  
  
SYNOPSIS  
#include <sys/types.h>  
#include <sys/wait.h>  
  
pid_t wait(int *wstatus);  
  
pid_t waitpid(pid_t pid, int *wstatus, int options);  
  
int waitid(idtype_t idtype, id_t id, siginfo_t *infop, int options);  
/* This is the glibc and POSIX interface; see  
NOTES for information on the raw system call. */  
  
Feature Test Macro Requirements for glibc (see feature\_test\_macros\(7\)):  
  
waitid():  
Since glibc 2.26: _XOPEN_SOURCE >= 500 ||  
_POSIX_C_SOURCE >= 200809L  
Glibc 2.25 and earlier:  
_XOPEN_SOURCE  
|| /* Since glibc 2.12: */ _POSIX_C_SOURCE >= 200809L  
|| /* Glibc versions <= 2.19: */ _BSD_SOURCE  
  
DESCRIPTION  
All of these system calls are used to wait for state changes in a child of the  
calling process, and obtain information about the child whose state has  
Manual page wait(2) line 1 (press h for help or q to quit)
```

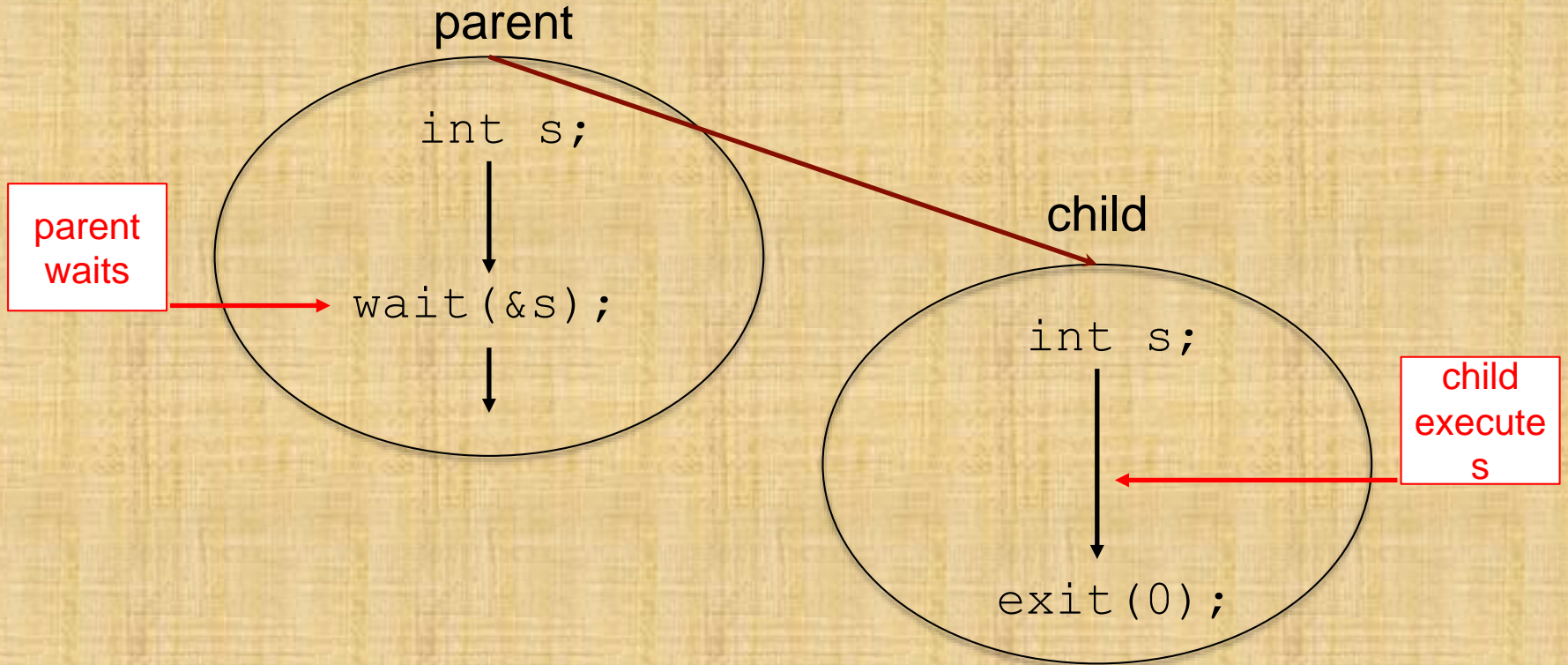
# Waiting

(the inverse of forking)



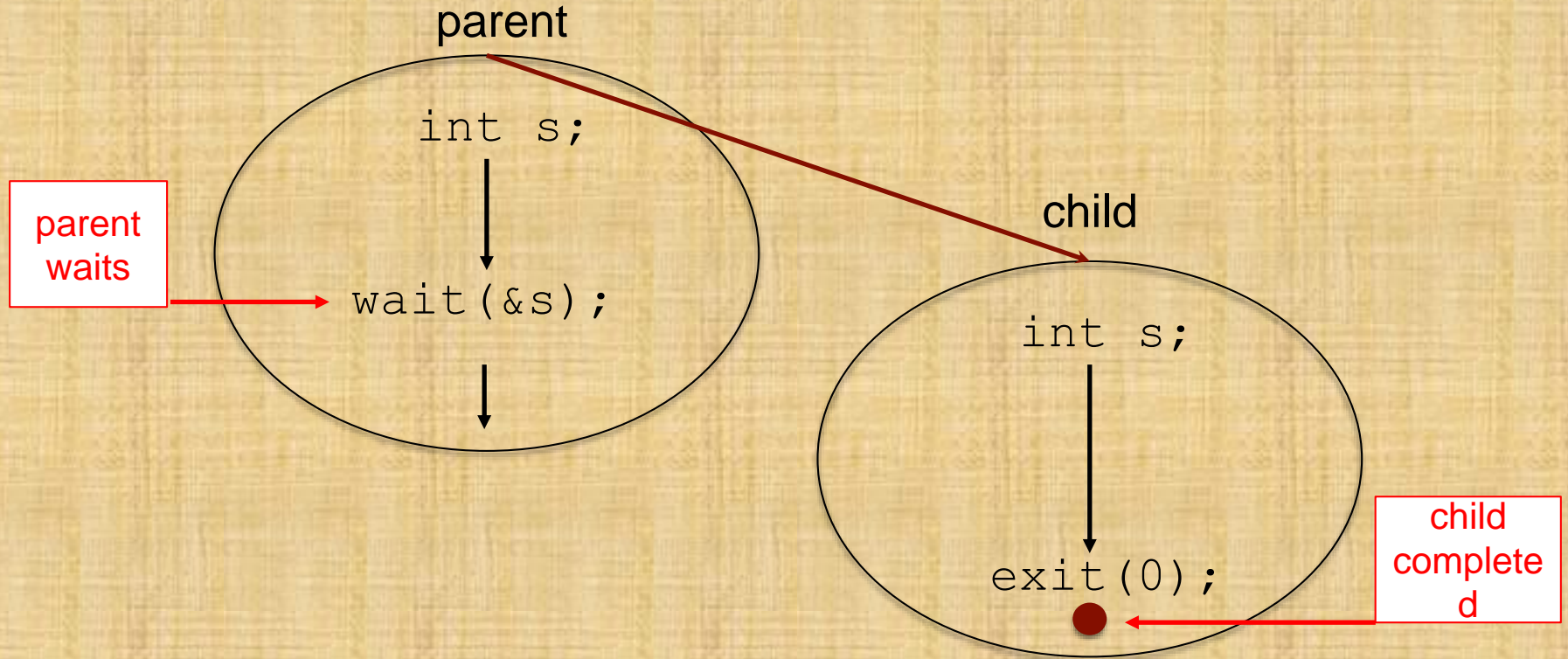
# Waiting

(the inverse of forking)



# Waiting

(the inverse of forking)



# Waiting

(the inverse of forking)

parent

```
int s;
```



```
wait(&s);
```



parent  
continues

