how to create instance?

fork() → new process . child process of parent process
execve() → run program within child process
code in kernel

in OS, all processes started from shell process
a running shell is a process

info can be inherited by parent shell like the pwd

shell has its own PCB

## Example

**Example:**

**Parent: the shell process**

**% ./myprog**

**Child: the myprog process**

**"PCB": process control block (i.e. proc states)**

**Flow of execution:**

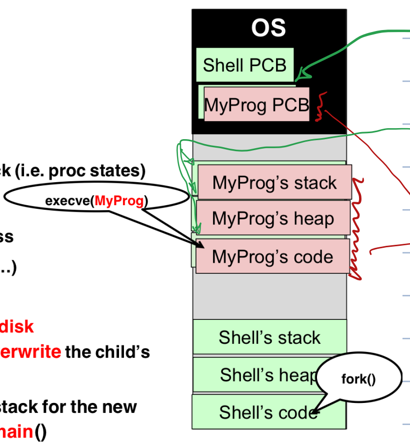
① **fork() create a child process**

**copy process state (PC, …)**

**execve(myprog)**

**Take the program from disk (/code/myprog) and overwrite the child's code segment**

**Initialize the heap and stack for the new program, and jump to main()**



① copy all, put here
create snapshot of parent program

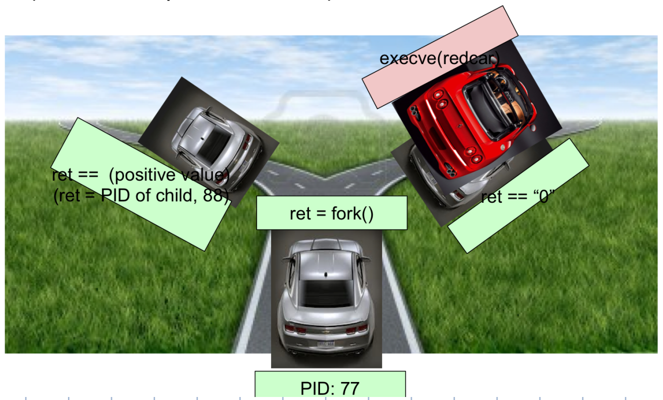
② in code, take program from disk & overwrite
reinitialize heap & stack

OS provides fork() & execve()
is a parent-child relationship
child inherits same process state of parent

all children will return 0 from fork()
parents return a pos. value
each can have their own process id getpid()

now within child process, run execve()
have own code, heap, stack seperate from parent

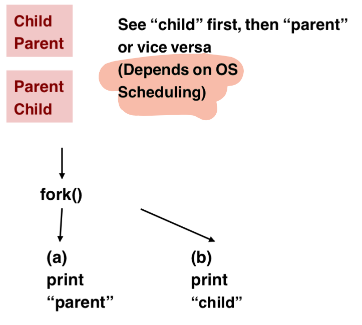
(the road is the sequence of instructions)



why 2 calls?

now, how to parallelize processes? → concurrency

```
void fork_ex9() {
    int ret = fork();
    if (ret == 0) {
        printf("Child");
    }
    else {
        printf("Parent");
    }
    exit();
}
```


Child
Parent
Parent
Child
See "child" first, then "parent" or vice versa
(Depends on OS Scheduling)
fork()
(a) print "parent"
(b) print "child"

(a) and (b) are concurrent

wait( ) → complete children processes first before parent
↳ not concurrent

Talking to eachother!

OS catches signals

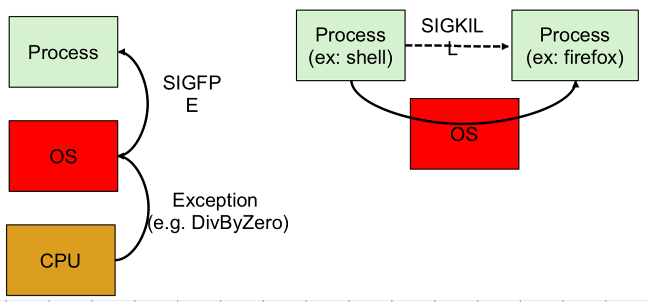
every program has a SIGFPE, signal handler
specifies how app deals w/ specific signals — unrelated to exceptions
↳ written by app dev.

Signal - small "message" notifies process that event of some type has occurred

A *signal* is a small "message" that notifies a process that an event of some type has occurred in the system

- Signal type is identified by small integer IDs (1-30)
- Signals can come directly from the OS
- Signals can come from other processes (mediated by the OS)

from OS or some other process

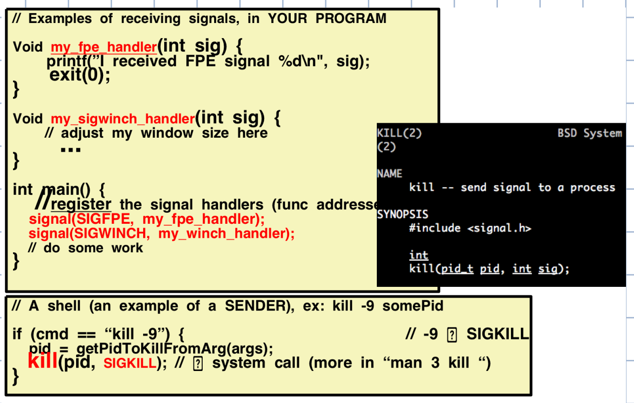


400

31

register signal handlers

kill sends signal to process



```
// Examples of receiving signals, in YOUR PROGRAM

Void my_fpe_handler(int sig) {
     printf("I received FPE signal %d\n", sig);
     exit(0);
}

Void my_sigwinch_handler(int sig) {
     // adjust my window size here
       ...
}

int main() {
   //register the signal handlers (func addresse
   signal(SIGFPE, my_fpe_handler);
   signal(SIGWINCH, my_winch_handler);
   // do some work
}
```

```
KILL(2)                BSD System
(2)

NAME
     kill -- send signal to a process

SYNOPSIS
     #include <signal.h>

     int
     kill(pid_t pid, int sig);
```

```
// A shell (an example of a SENDER), ex: kill -9 somePid

if (cmd == "kill -9") {                          // -9 ⍰ SIGKILL
   pid = getPidToKillFromArg(args);
   kill(pid, SIGKILL); // ⍰ system call (more in "man 3 kill ")
}
```