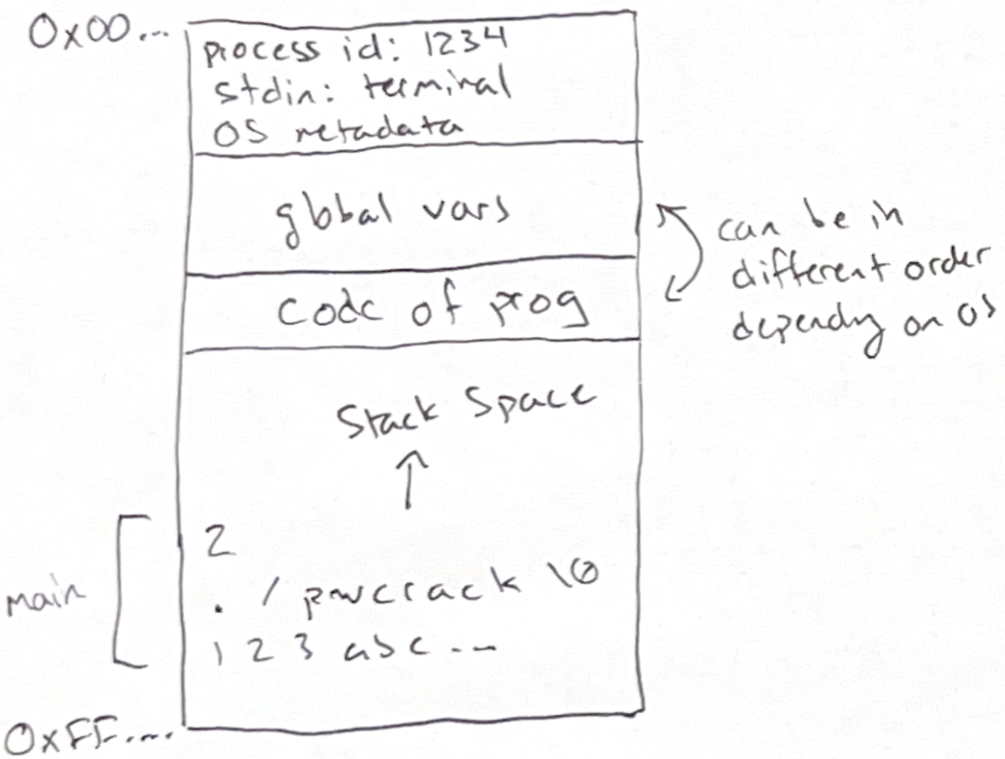
What happens when you run a program?

$ ./pwcrack 123abc...

Your OS starts a process: a running program



can be in different order depending on OS

Vocab:

pid = process id
  unique id OS uses to track processes

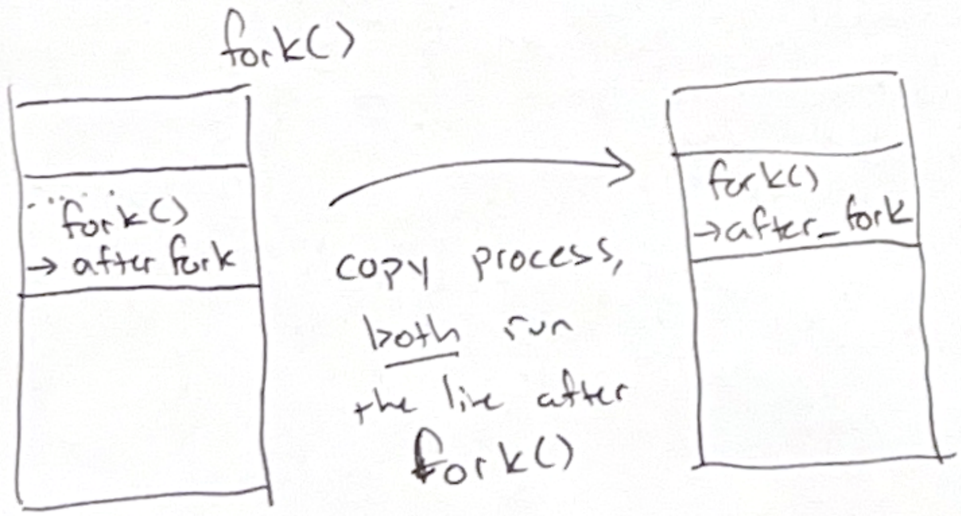
address space
  the memory given to process by the OS

section (of address space)
  OS, global, code, stack

int fork() "system call"

makes a copy of the current process
- new process starts running immediately
- starts running from the same line in the code
  (the line right after fork())
- the "parent" process also keeps running!
- returns 0 in the child process
- returns the pid of the child process in parent

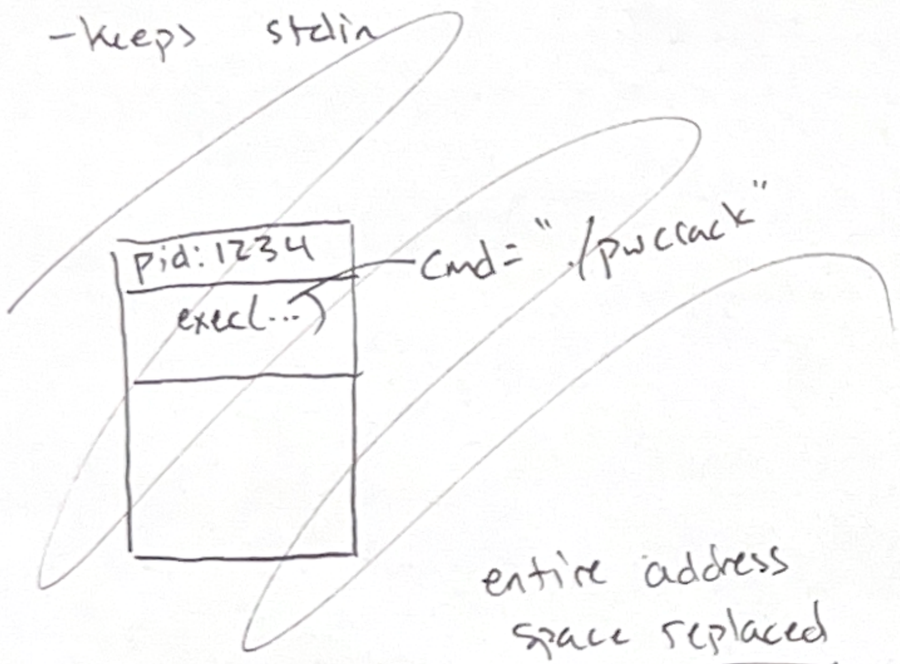
```
int pid = fork();
if (pid == 0) { ... do child process stuff ... }
else { ... parent process continues ... }
```



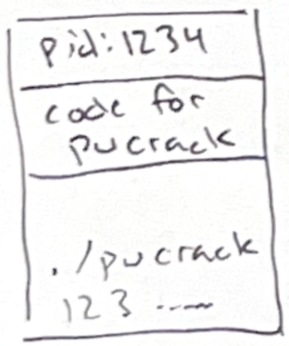
void execvp(char * cmd, char ** args)

replaces the current process with a newly-started process based on the given cmd and args (just as if you started from command-line)

- keeps pid
- keeps stdin



entire address space replaced



Imagine you are implementing Finder
Windows Explorer

```
void doubleclick(FileIcon icon) {
    int pid = fork();
    if (pid == 0) {
        execvp(icon.progname, icon.args);
    } else {
        highlightIcon(icon);
    }
}
```

"chrome"
"spotify"

```c
#include <string.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>

#define CMD_LENGTH 1000

// Puts arguments into result using strtok
int parse_args(char* cmd, char** result) { /* from Wednesday */ }

int main() {
  char cmd[CMD_LENGTH];
  char* args[CMD_LENGTH];

  while(1) {
    printf("$ ");
    char* result = fgets(cmd, CMD_LENGTH, stdin);
    if(result == NULL) { break; }
    cmd[strcspn(cmd, "\n")] = '\0';

    int argc = parse_args(cmd, args);
    args[argc] = NULL;



    if(                                          ) {
      printf("fork() failed: %d\n", pid);
      break;
    }

    else if(                                        ) {



    }
    else {



    }

  }
}
```

```
$ gcc -Wall shell.c -o shell
$ ./shell
$ ls -l
Started ls as pid 65819
total 24408
-rw-r--r--@ 1 joe  staff       706 Oct 24 09:21 shell.c
...
65819 finished
```

- `int fork()`

  Starts a new process that is an exact copy of this one. Returns 0 in new process and the process id of the new process to the parent process.

- `void execvp(char* program, char** args)`

  Starts the given program with the given arguments, replacing current process.

- `void waitpid(int pid, int* status, int options)`

  In a parent process, waits for the process with the given pid to finish. (You can pass `NULL` for `status` in programs that don't care how it terminated, and `0` for `options` to get the defaults which are fine for our purposes. Use `man waitpid` for more!)