Today: execvp

when invoked from process: gut existing process & rebuild from new executable
↳ supplied from *path

returns -1 if execution fails

does not return if it succeeds - b/c the program running at that pt. no longer exists

program pastes global data segment w/ its own code

new heap, new stack frame for main

First example: execvp + fork + waitpid as a duo (which is common)

```
static void mysystem(char *command); //don't want to kill off terminal session when command is issued
                                     //writing REPL (read-eval-print loop)

int main(->) {
    while(true) {
        printf("> ");
        char command[2048];
        fgets(command, 2048, stdin);
        if (feof(stdin)) break; //if I've shut down this file* with control+D - "never going to get any data again"
        command[strlen(command)-1] = '\0'; //chomp
        mysystem(command);
    }
    printf("\n");
    return 0;
}
```

```
static void mysystem(char *command) {
    pid_t pid = fork();
    if (pid == 0) {
        char *argv[] = {command, NULL}; //replace w/ {"/bin/sh", "-c", command, NULL}
        execvp(argv[0], argv);
        printf("Fail\n");
        exit(0); //cannot be return so we know to "clean up" this shell (details in lecture recording)
    }
    waitpid(pid, NULL, 0);
}
```

this won't work for e.g. "make clean" - parsing issue

verbatim what should be passed to the main fxn of the executable

now I know I'm in the parent

trailing ampersand means run this in the background, giving you the prompt immediately back

```
simplesh > sleep 10 &
```

↓

//simplesh.c

```
int main(->) {
    while (true) {
        printf("simplesh> ");
        char command[2049];
        readCommand(command, 2048); //wrapper for the fgets() stuff
        char *argv[128]; //idea: parse the command buffer to tokenize into arguments vector
        int count = parseCommand(command, argv, 128);
        if (count == 0) continue;
        if (strcmp(argv[0], "quit") == 0) break;
        bool isBackground = strcmp("&", argv[count-1]) == 0;
        if (isBackground) argv[--count] = NULL; //you don't need the ampersand - just strip it
        pid_t pid = fork();
        if (pid == 0) execvp(argv[0], argv);
```

very common to invoke execvp as part of a child process

```
if (isBackground) printf("%d %s\n", pid, argv[0]);
else waitpid(pid, NULL, 0);
```

Next example: usage of execvp w/o fork

```
$ printf "144 255" | xargs factor 100
```

→ will run "factor 100 144 255"

↑ read into stdin the stuff printed to stdout on other side of pipe

```
int main(int argc, char *argv[]) {
    vector<string> tokens;
    pullAllTokens(cin, tokens);
    char *xargv[argc + tokens.size()];
    for (int i=0; i<argc-1; i++)
        xargv[i] = argv[i+1];
    for(int i = argc-1; i<argc-1+tokens.size(); i++)
        xargv[i] = tokens[i-argc+1].c_str();
    xargv[argc+tokens.size()-1] = NULL; // built new argv vector with new argv[0]... preamble to just running the executable!
    execvp(xargv[0], xargv);
}
```

wednesday: more elaborate (program mgmt.) than waitpid, etc.