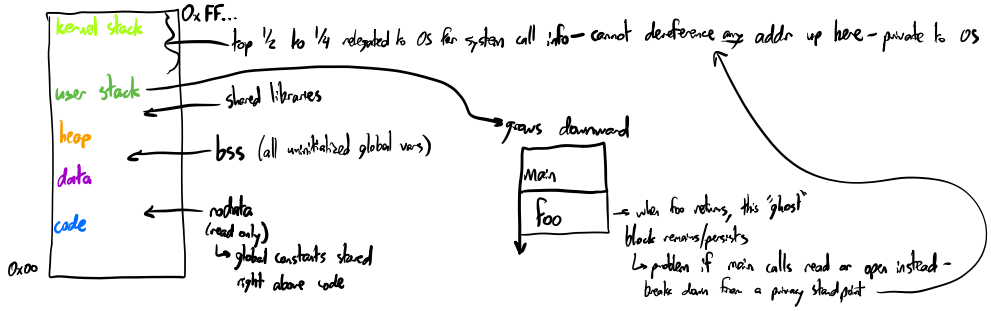
today: system call fns

so core to execution of OS, have to execute w/ "superhuman" permissions

open/read/write/close

each process operates as if it has all of main memory

64-bit addr means $2^{64}-1$ bits to address



now... creating new processes (instead of files)

fork

```
int main(->) {
    printf("Hello\n");
    fork(); //creates second copy of process calling fork, allow both to continue in parallel
    printf("Goodbye\n");
    return 0;
}
```

```
const char * kTool = "abcd";
int main(->) {
    for (size_t i=0; i< strlen(kTool); i++) {
        printf("%c\n", kTool[i]);
        pid_t pid = fork();
    }
    return 0;
}
```

← will continue execution from the middle of for loop — everything is cloned at time of fork

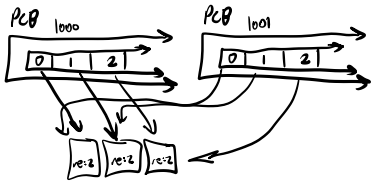
→ b/c process created in middle of fork() call

→ returns process ID of child in original

returns 0 in child — easy way to differentiate

process ID type: special kind of int

how are they all printing to the same terminal? Fork creates copy of entire PCB of parent process, including aliases to file descriptor table



fork bomb: while(true) fork();

takes total ownership of CPU

another system call:

```
pid_t waitpid(pid_t pid,
              int * status,
              int options);
```

```
int main(->) {
    printf("Hello\n");
    pid_t pid = fork();
    if (pid == 0) {
        //we are in the child of the forking process
        printf("I'm the child, now waits for me\n");
        return 110;
```

```
    } else {
        int status;
        waitpid(pid, &status, 0);
        printf("Child returned %d. I'm quitting.\n", WEXITSTATUS(status));
    }

    return 0;
}
```