today

fork

waitpid

execvp

how new executables introduced into system

from wednesday: fork() → returns 0 for child b/c child can just call getpid() for its own PID #

data structure related is deep, (memory) independent but identical clone

file descriptors, assembly code instruction identical

fork() itself is not a blocking system call → talking about "time slice" a process is alotted on a CPU

waitpid

parent process waits for child

only direct parent, not grandparent

waitpid(pid, &status, 0);

↳ int status is a 32-bit vector incl. things like exit status

↳ WEXITSTATUS(status);

another example (how deep the clone is):

```
int main() {
    printf("one time.\n");
    pid_t pid = fork();
    bool parent = pid > 0;
    if ((random() % 2 == 0) == parent) {//random num generates same exact # in parent & child! seed is the same (replicated in fork call)
        sleep(5);
        printf("I love naps.\n");
        return 110;
    }
    printf("I'm the parent.\n");
    if (parent) waitpid(pid, NULL, 0);
    return 0;
}
```

↳ see example code in lecture slides (this code is slightly wrong - wouldn't be the parent necessarily printing "I'm the parent")

coin flip - who gets to nap, parent or child?

```
int main(...) {
    for (size_t i=0; i<8; i++) {
        pid_t pid = fork(); //not storing pids anywhere!
        if (pid == 0) return 110+i;
    }
    for(size_t i=0; i<8; i++){
        pid_t pid = waitpid(-1, &status, 0);
        printf("Child %d exited
            with code %d.\n",
            pid, WEXITSTATUS(status));
    }
```

waitpid returns pid of child that returned

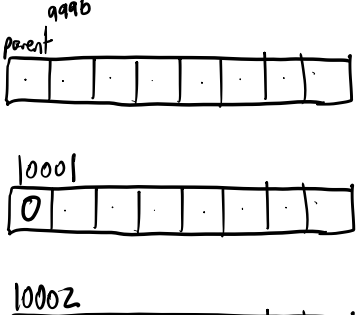
-1 says wait for any child to finish

waitpid is what deallocates process control block

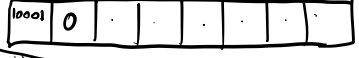
"reaping"

now, keep track of children spawned in order!

```
int main(...) {
    pid_t children[8];
    for (size_t i=0; i<8; i++) {
        children[i] = fork();
        if (children[i]==0) return 110+i;
    }
    for (size_t i=0; i<8; i++) {
        int status;
```

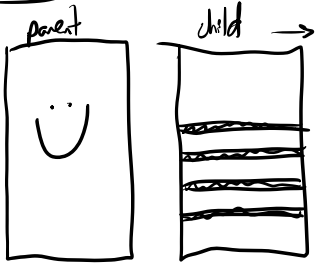


```
        waitpid(children[i], &status, 0);
        printf("%d %d\n", children[i], WEXITSTATUS(status));
    }
    return 0;
}
```



this is "child code" - but what if we instead want to reference an executable?

execvp



→ child is completely "cannibalized" w/ new stack, heap, etc. & entry into executable's main func.

int execvp(const char *pathname, char *argv[]);

returns -1 if executable fails

does not return if executable succeeds - b/c everything has been killed off incl. call to execvp