CS 3650 – Computer Systems
Spring 2024
Peter Desnoyers

Lecture 6, Thur Jan 25 2024

Yet more C stuff

variable | expression | value

int x; ← definition — x is an int, create space for x

x = 1; ← assignment

x+1 x==2 (1+2 (x+y)*z

expression ⟶ has a value

* — dereference (value pointed to)

& — address of

x == *(&x)

x: 

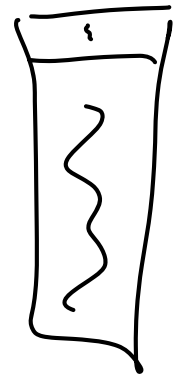
* works on bigger things (eg struct)

```
struct b100{
  int a;
  char b[96];
};
```

```
struct b100 X;
```

&X

X.a
X.b[10]



```
struct b100 *p = &X;
```

p->a
p->b[10]

≡

(*p).a
(*p).b

p

struct b100 *

struct b100 (sizeof(*p) == 100)

Back to fork, exec (+ dup2 & waitpid)

```
int pid = fork()
if (pid == 0)
    .. child
else
    .. parent
```

```
execvp("cmd", char** argv)
```

use path

takes argv array

python:

```
import sys
sys.argv[]
len(sys.argv)
```

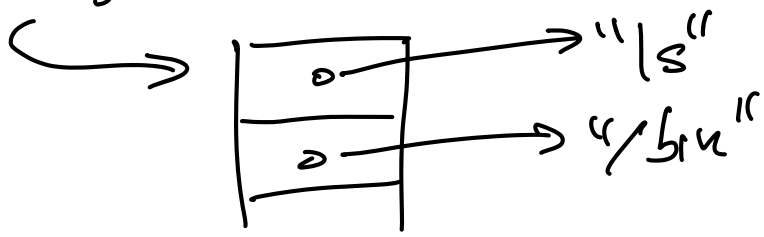
argv[0] ← command name

ls /bin ⇒ ("ls", "/bin")

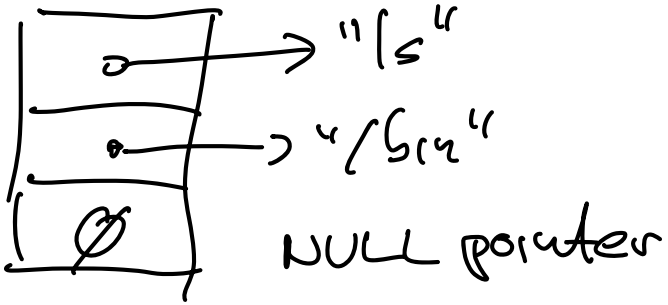
C: main(int argc, char **argv)

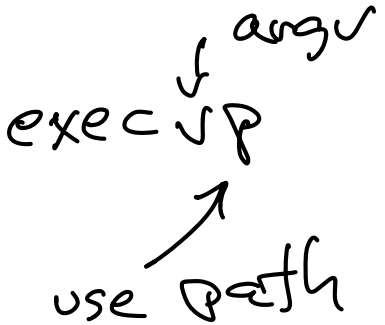
argc arguments (≥ 1)

argv : array of char *



execvp - where's argc?





command path

echo $PATH

/dir1:/dir2: ... :/usr/bin

execvp("ls", argv)

/dir1/ls ? no

/dir2/ls ? no

... /usr/bin/ls ? yes

error

main() {

:

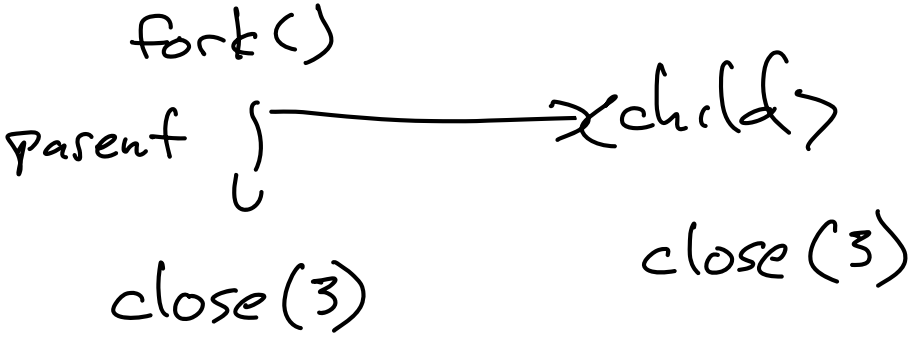
new program

... exit()

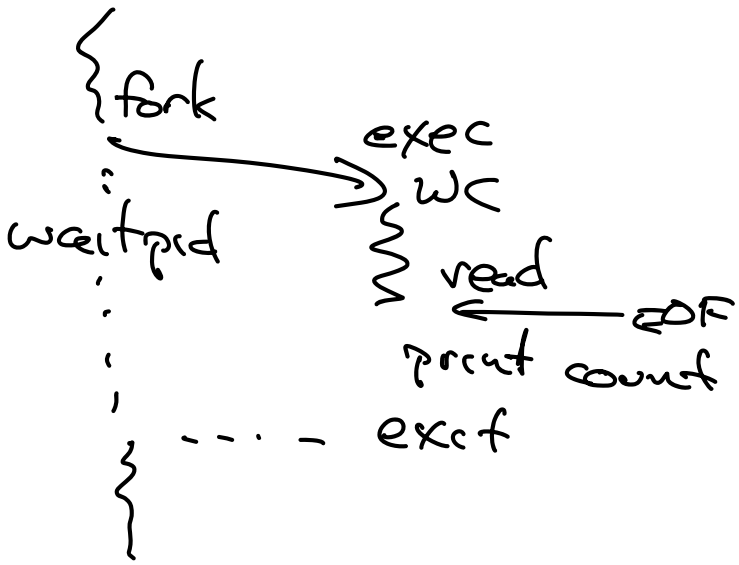
exec doesn't wipe everything (open files)

```
int fd = open("file.txt", RDONLY)
```

← 3




```
int pid = fork()
if pid == 0
    exec("wc")
else
    waitpid(pid, ...)
```



How do we redirect I/O?

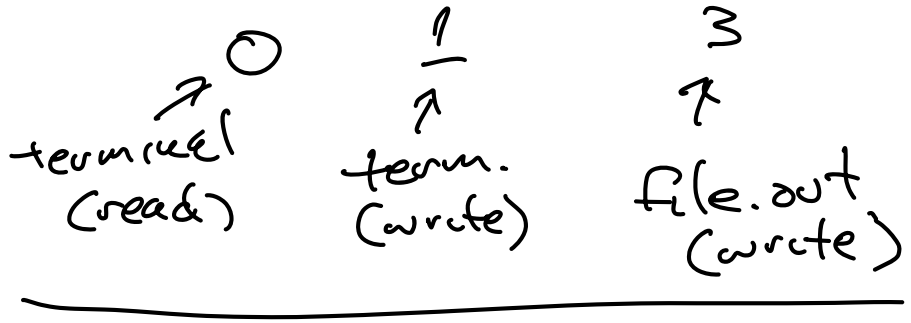
ls > file.out wc < file.txt

```
fd = open(file.out, O_WRONLY | ...)
dup2(fd, fd2)
dup2(fd, 1)
close(fd)
```

3

3

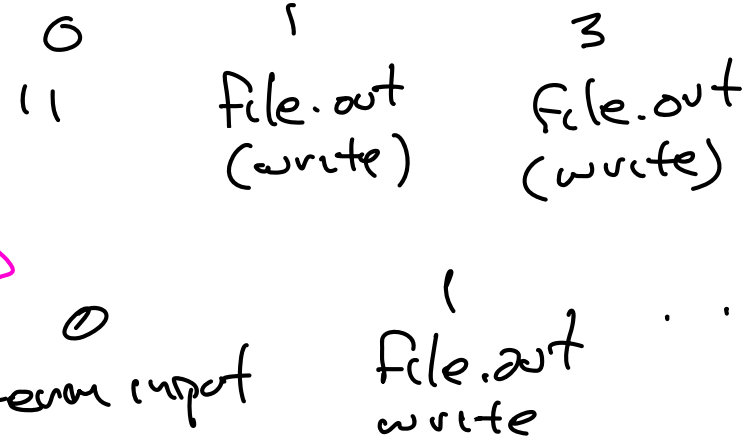
copy



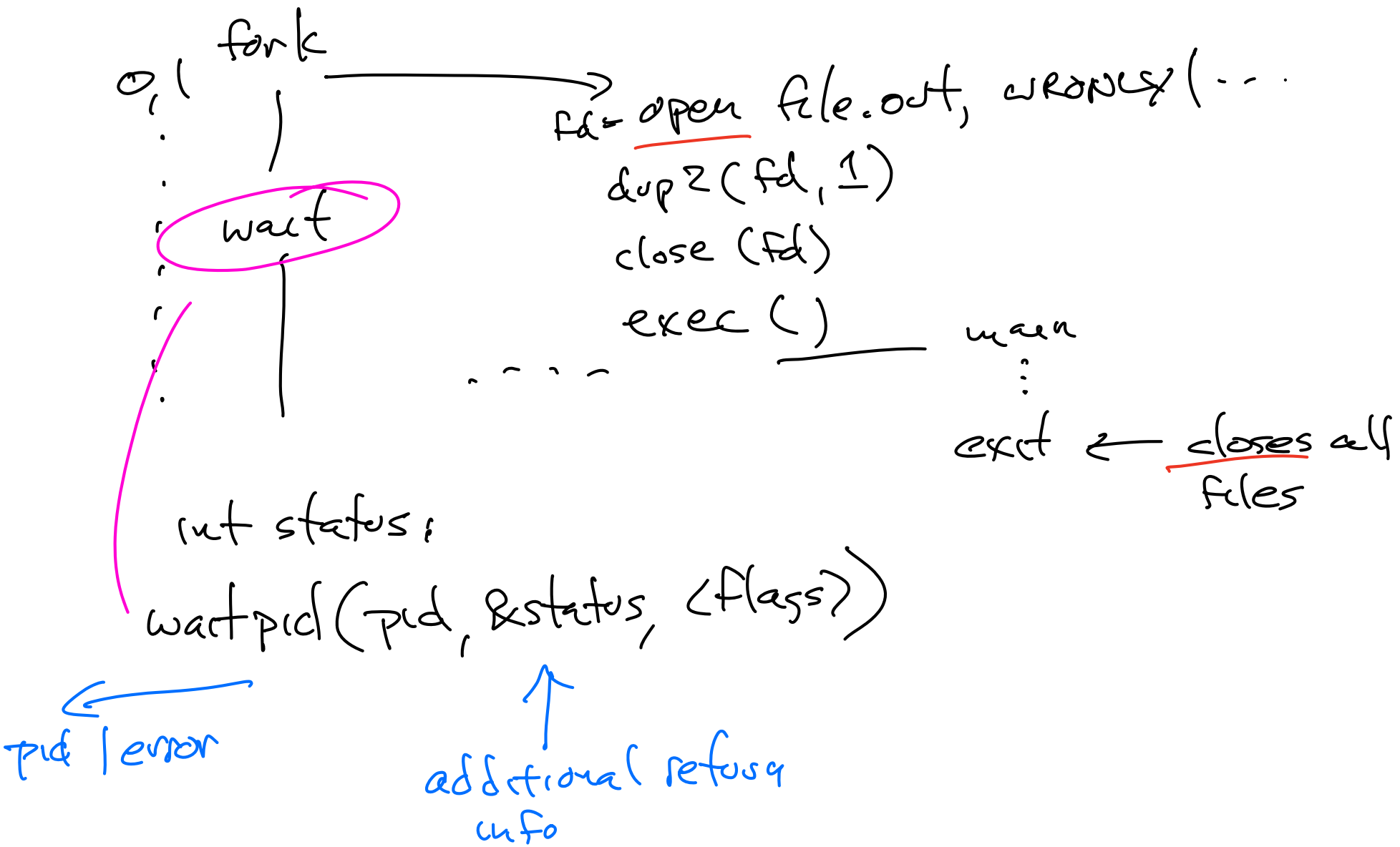
dup2(3, 1)

closing 1

duplicating 3 → 1



when do we do it?



```
for f in file1 file2 file3; do
    if grep lastfile $f; then
        echo $f is last
        break
    fi
done
```

command

truth value:

true success

false error

true.c

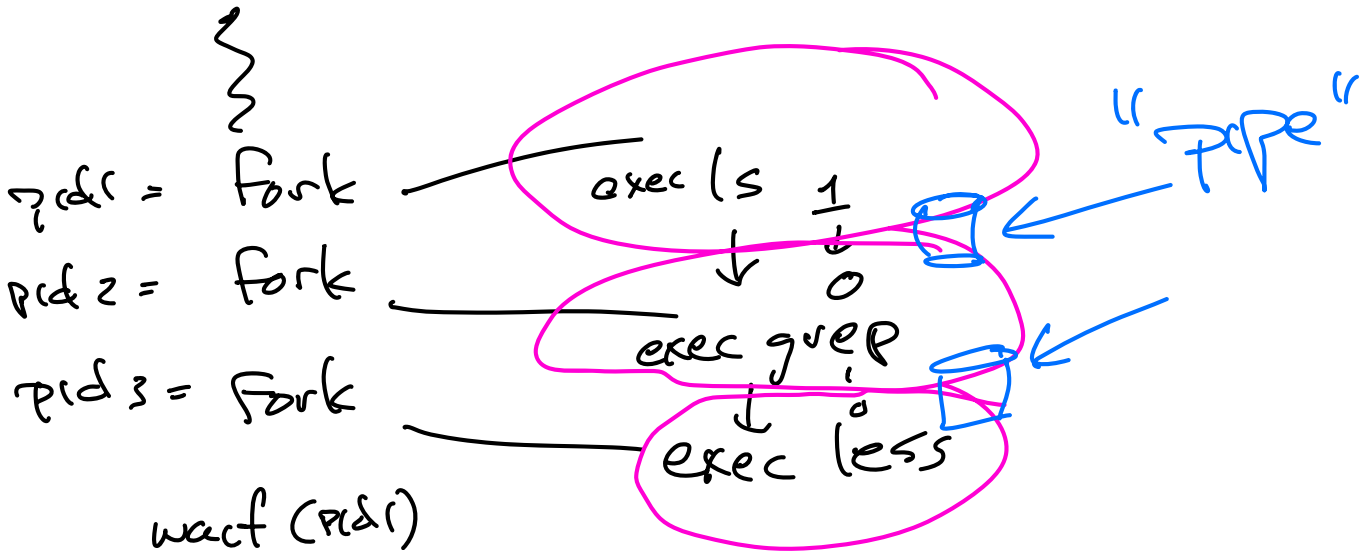
```
main(){
    exit(0)
}
```

false.c

```
{
    exit(1)
}
```

Pipes

ls | grep zip | less



wait (pid1)
wait pid2
wait pid3

int pipe_fds[2]

pipe (pipe_fds)

pfds[0] ← read end

pfds[1] ← write end

parent

pfd[0], pfd[1]

after fork

close p[0]

close p[1]

child 1

dup → 1

pfd[1] output

child 2

pfd[0] input

↳ dup → 0

close

signals

kill <pid#>

kill(pid)