Syscalls: functions that ask the OS to do something we couldn't do ourselves
return −1 on error

I/O syscalls:

* int open(const char* filename, int flags, ...);
↳ Session ID ("file descriptor")
Info about what you want to do

ssize_t read(int fd, void *dest_buf, size_t count);
↳ how many bytes were read into buf

ssize_t write(int fd, void *src_buf, size_t count);
!! not guaranteed to write all 'count' bytes

Not in this class: int lseek(int fd, ...) / int stat

* int close(int fd);

Open flags:
- O_RDONLY: read only
- O_WRONLY: write only
- O_CREAT: create file if it doesn't exist
- O_EXCL: raise an error if file already exists

File permissions

read, write, execute

RWX
110
$2^2\ 2^1\ 2^0$
4 2 1
$= 2+4 = 6$

RWX
101
$1+4=5$

$4 = R$

$6 \overset{(4+2)}{=} RW$

$7 \overset{(4+2+1)}{=} RWX$

$3 \overset{(2+1)}{=} WX$

---

3 digits: owning user, owning group, everyone else

$$\begin{tikzcd}
0 \arrow[d] & 6 \arrow[d] & 4 \arrow[d] & 4 \arrow[r] & \text{everyone else can R} \\
\text{Octal number} & \text{owner can RW} & \text{group can R} & &
\end{tikzcd}$$

What if we want RW for user, R for group, no access for everyone else?

0640

Error handling   int errno;

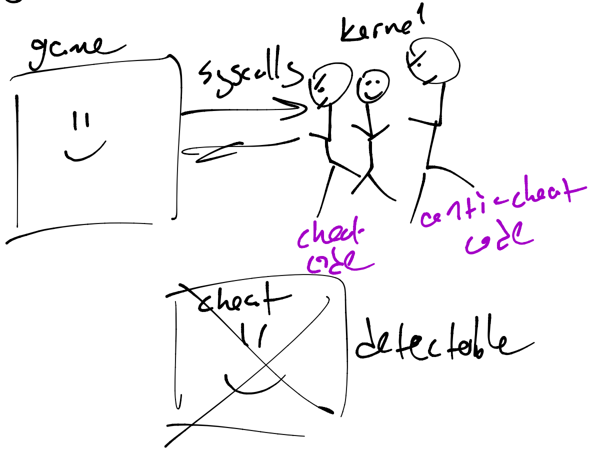
Syscalls return -1 if something bad happens

set errno to some number indicating cause of failure

perror(const char * message); look at errno and print a human readable string

Key points for syscalls:

- Syscalls are how you interact with the "outside world"
- open is like malloc, close is like free
- Any time you call a syscall, check for errors

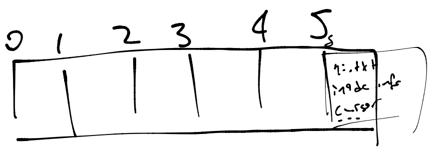


File sessions

program 1

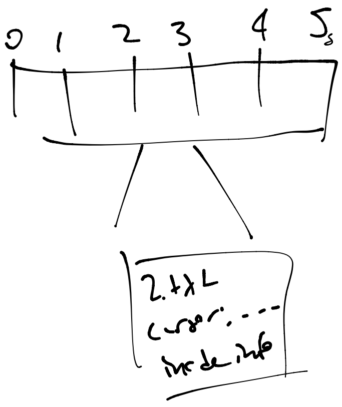
open("hi.txt", ...) → 5

read(3, ...);



program 2:

open("2.txt", ...) → 3



kernel: hypothetical scenario:

{
5:
- "hi.txt"
?? - cache inode info
- cursor: 0

3:
- "2.txt"
- inode info
- cursor

4:
- "hi.txt"
- cursor: 0
?? - cache inode info
}