System Level I/O

CS**351** : Saelee

UNIX I/O

everything is a file

regular files

text & binary

directories

indices of other files

links

“reference” to a file

symbolic / hard

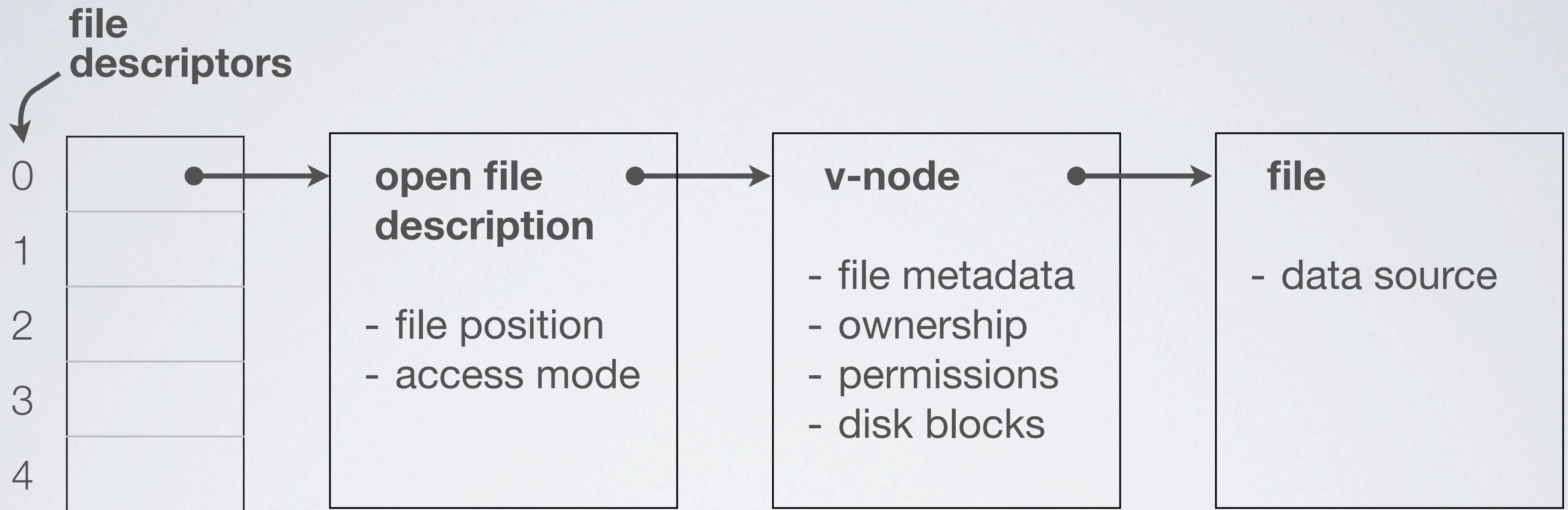
special files

character & *block* special

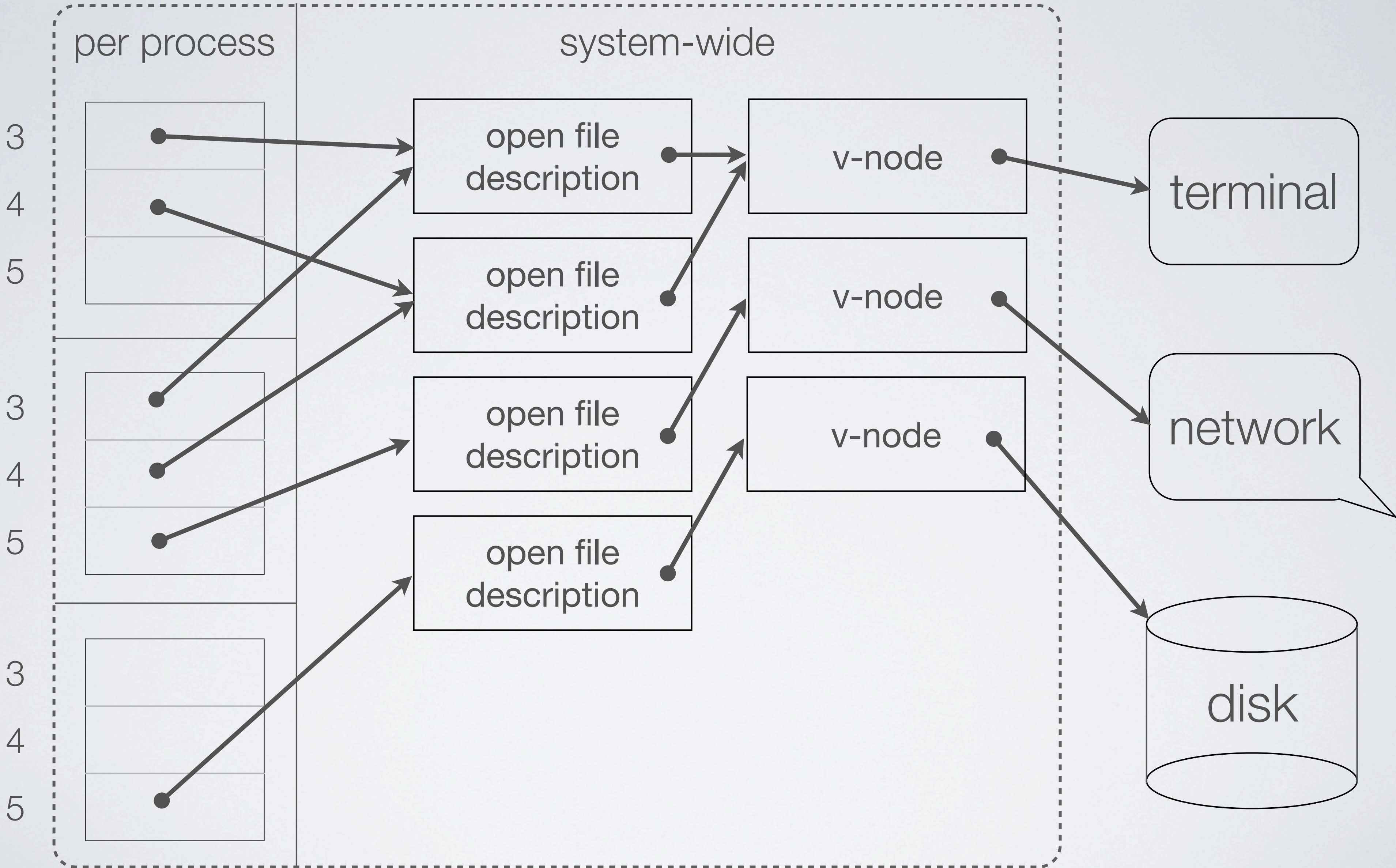
consistent access

```
open()
close()
read()
write()
lseek()
```

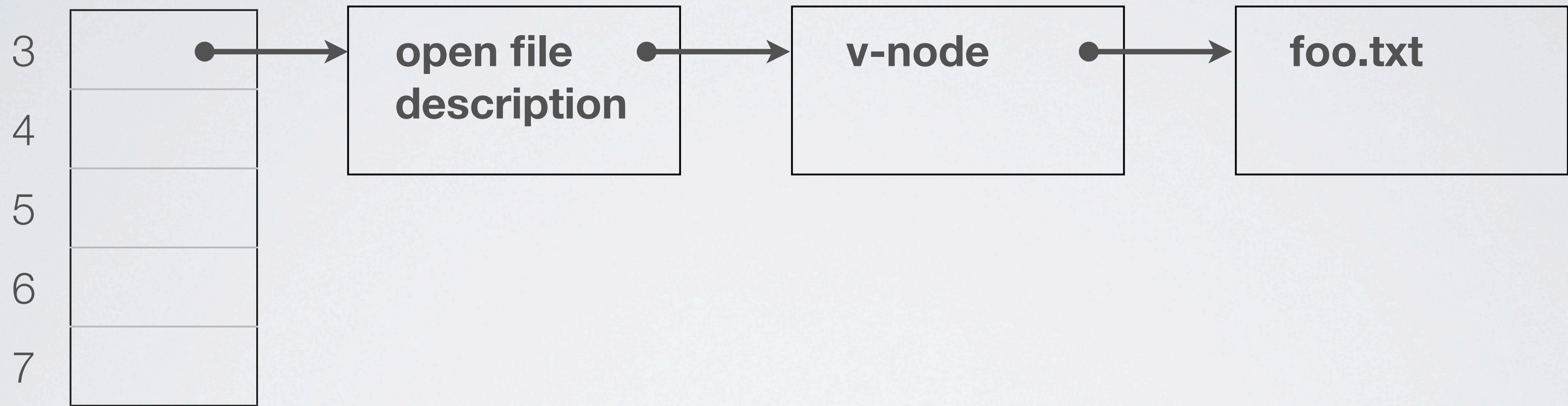
file descriptors


file
descriptors
0
1
2
3
4
open file
description
- file position
- access mode
v-node
- file metadata
- ownership
- permissions
- disk blocks
file
- data source

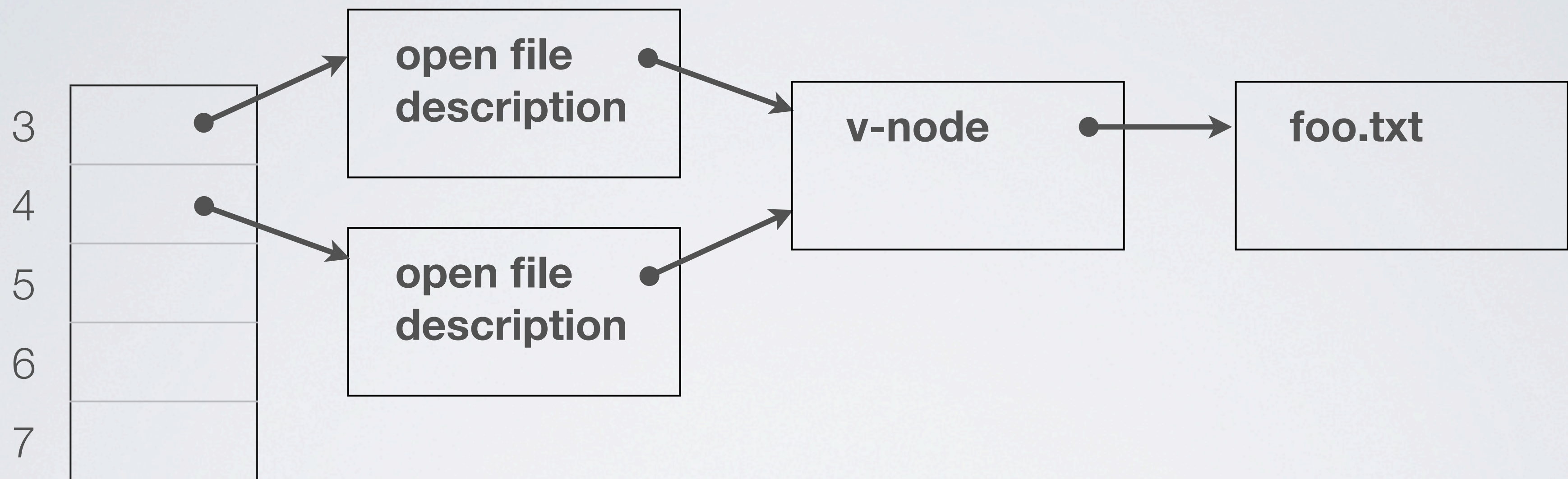
0 : standard input
1 : standard output
2 : standard error


kernel address space
per process
system-wide
3
4
5
3
4
5
3
4
5
open file description
open file description
open file description
open file description
v-node
v-node
v-node
terminal
network
disk

```
int open (const char *path,
          int oflag, ...);
```




```
open("foo.txt")
```



open("foo.txt")

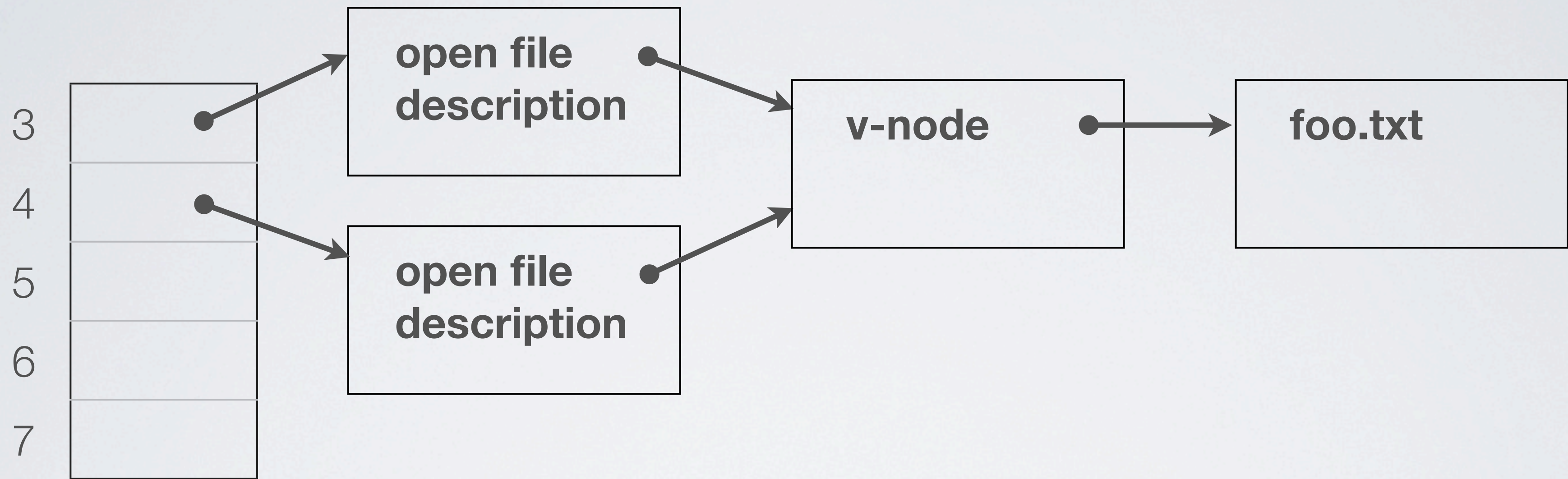
```
int fstat(int filedes,
          struct stat *buf);
```

```c
struct stat {
    dev_t         st_dev;      /* device */
    ino_t         st_ino;      /* inode */
    mode_t        st_mode;     /* protection */
    nlink_t       st_nlink;    /* number of hard links */
    uid_t         st_uid;      /* user ID of owner */
    gid_t         st_gid;      /* group ID of owner */
    dev_t         st_rdev;     /* device type (if inode device) */
    off_t         st_size;     /* total size, in bytes */
    blksize_t     st_blksize;  /* blocksize for filesystem I/O */
    blkcnt_t      st_blocks;   /* number of blocks allocated */
    time_t        st_atime;    /* time of last access */
    time_t        st_mtime;    /* time of last modification */
    time_t        st_ctime;    /* time of last status change */
};
```

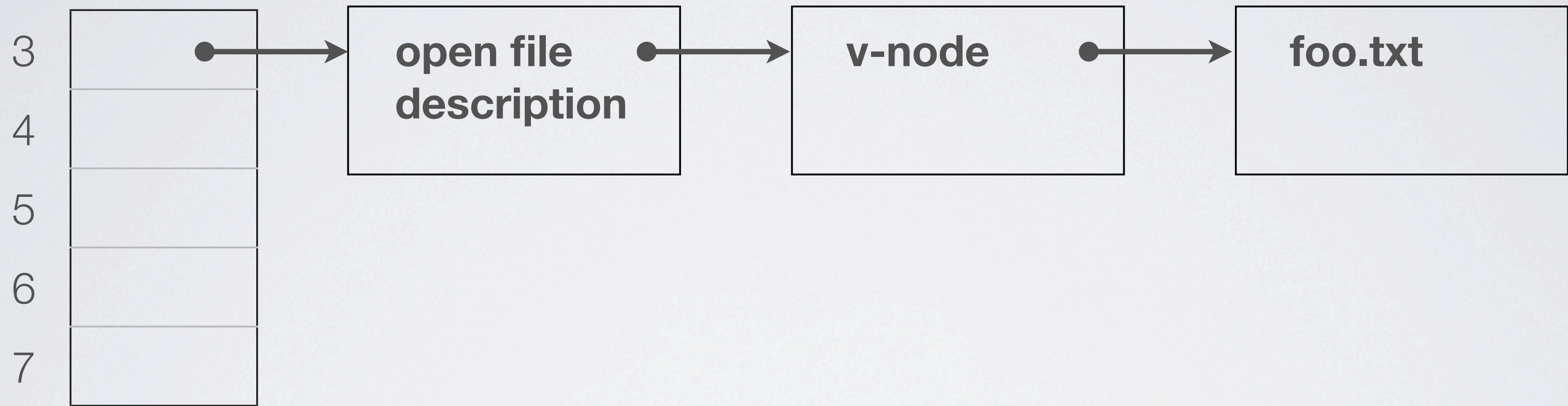
main.c:1 <No selected symbol>

```c
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <fcntl.h>

int main (int argc, const char * argv[]) {
    int fd = open("/bin/ls", O_RDONLY);
    struct stat info;
    fstat(fd, &info);
    return 0;
}
```

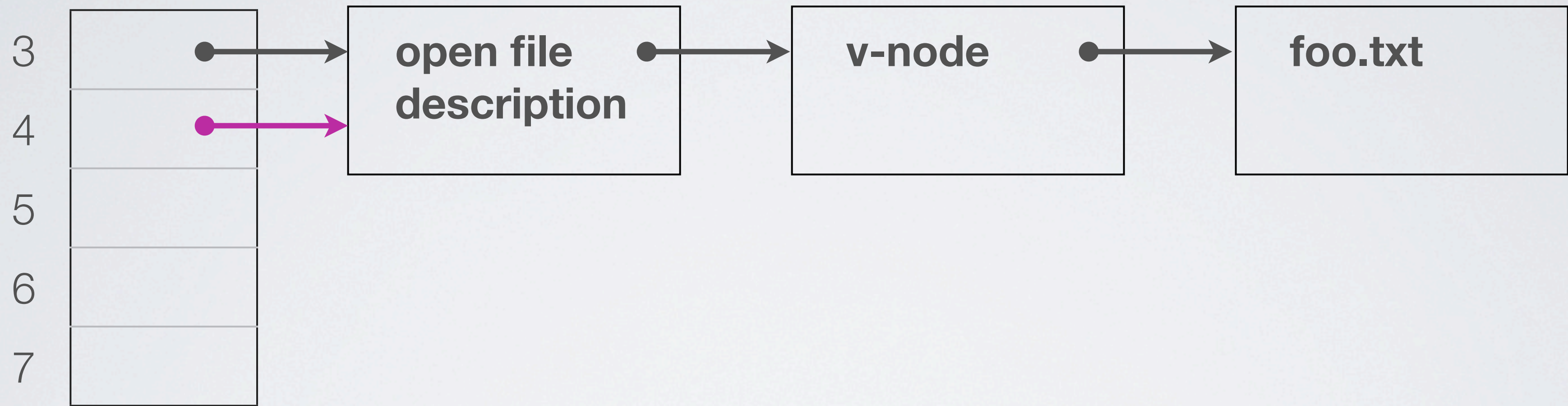
struct stat	info	{...} Summary
dev_t	st_dev	234881026
mode_t	st_mode	33133
nlink_t	st_nlink	1
__darwin_ino64_t	st_ino	3592124
uid_t	st_uid	0
gid_t	st_gid	0
dev_t	st_rdev	0
struct timespec	st_atimespec	{...}
struct timespec	st_mtimespec	{...}
struct timespec	st_ctimespec	{...}
struct timespec	st_birthtimespec	{...}
off_t	st_size	196504
blkcnt_t	st_blocks	384
blksize_t	st_blksize	4096
__uint32_t	st_flags	0
__uint32_t	st_gen	0
__int32_t	st_lspare	0
__int64_t [2]	st_qspare	[2]


3
4
5
6
7
open file description
open file description
v-node
foo.txt

```
int dup (int fd);
int dup2 (int fd1, int fd2);
```



open("foo.txt")

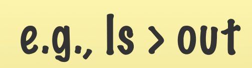


dup(3)

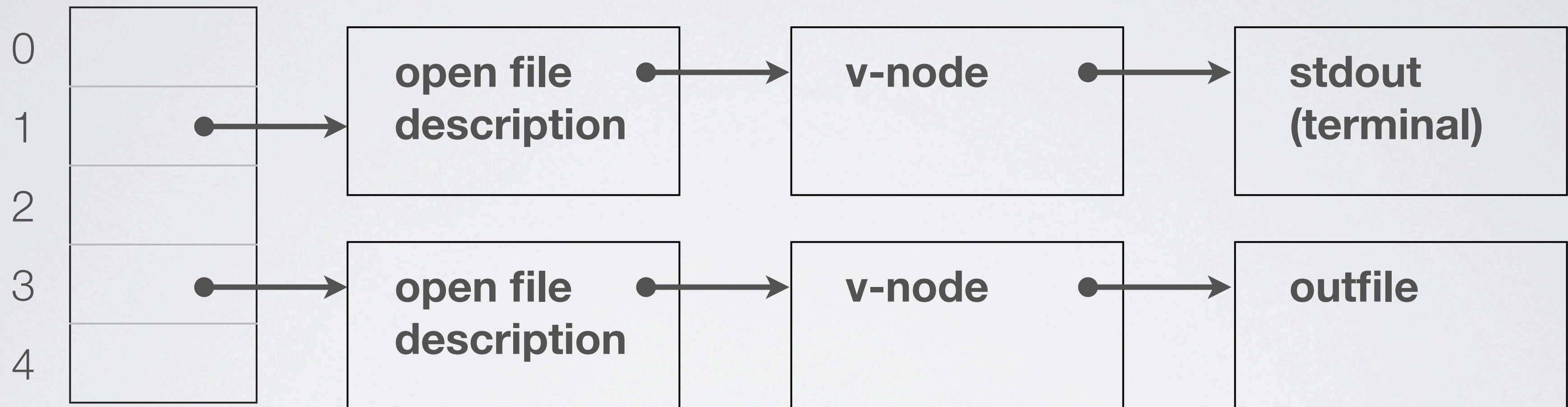
shared file position

I/O redirection

Demo

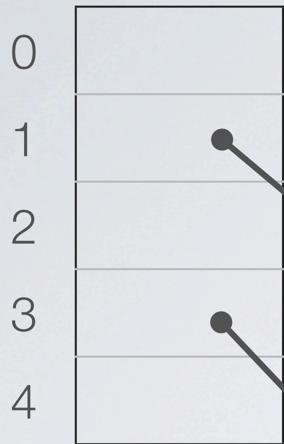


```
$ ls -la > lsout.txt

$ wc < my_novel.txt
    50210   2020392   3204860010

$ awk -F: {'print $1'} /etc/passwd | sort > users.txt

$ head -5 users.txt
abarua
adevalam
aflavell
akalanta
amanda
```



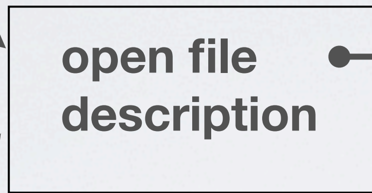

```
open("outfile")
```

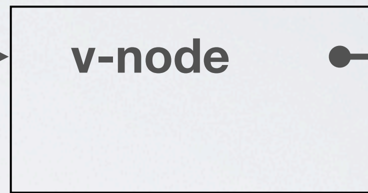
shell

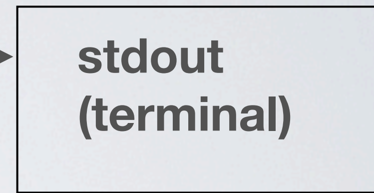


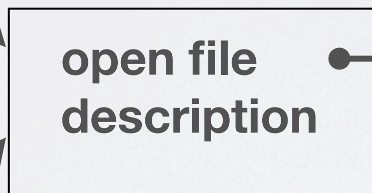
fork

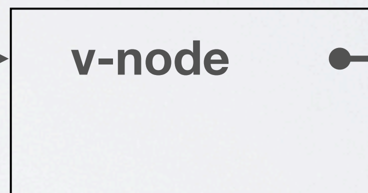
child

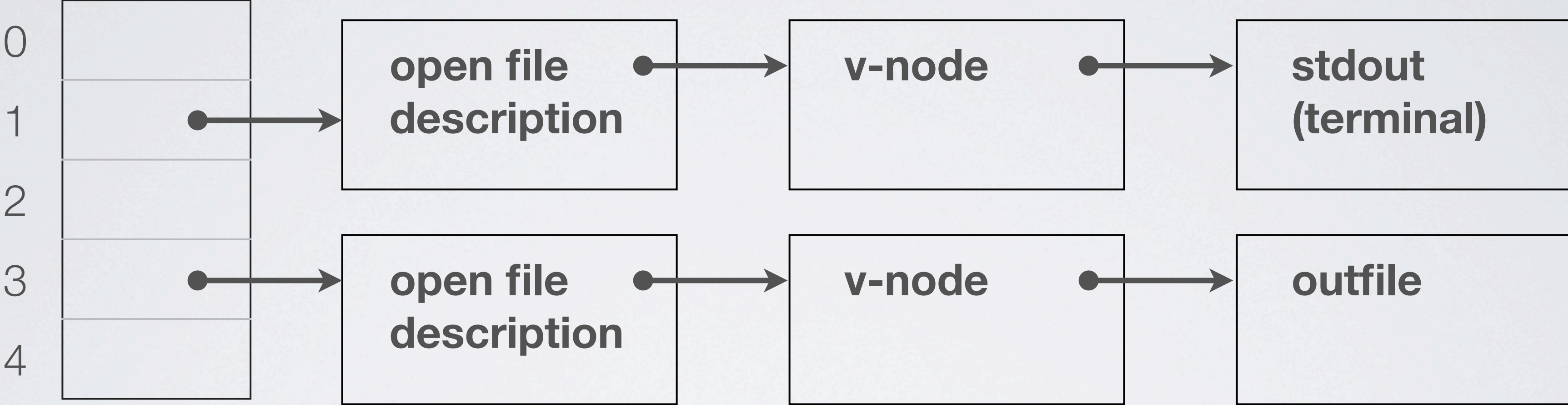




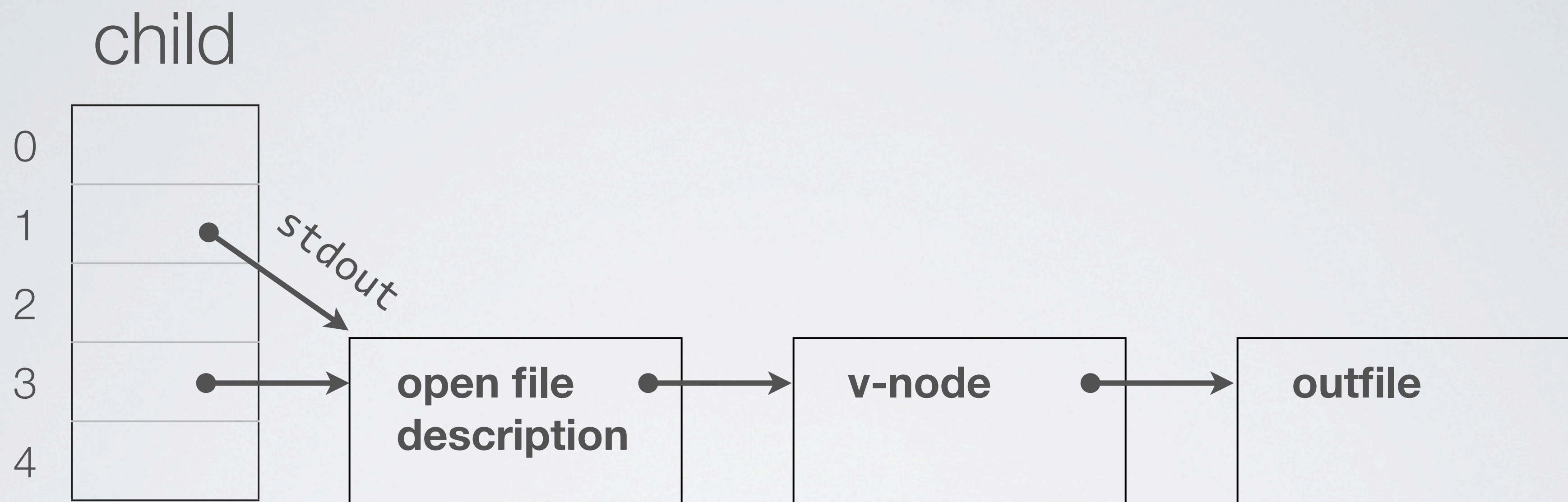




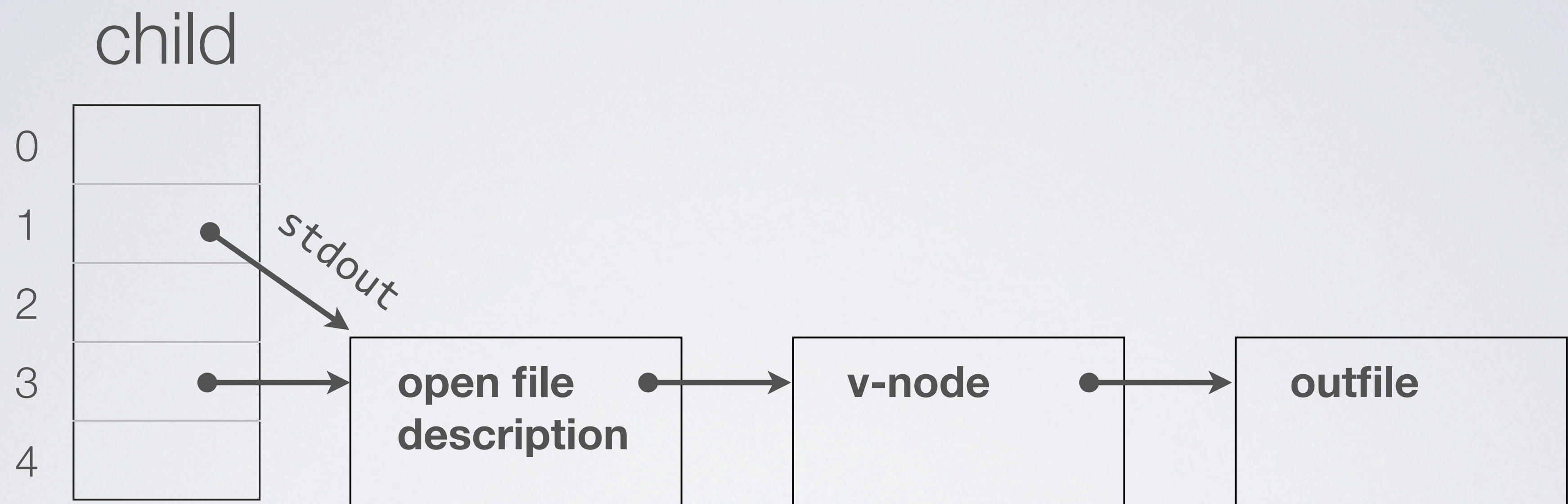




child
0
1
2
3
4
open file
description
v-node
stdout
(terminal)
open file
description
v-node
outfile

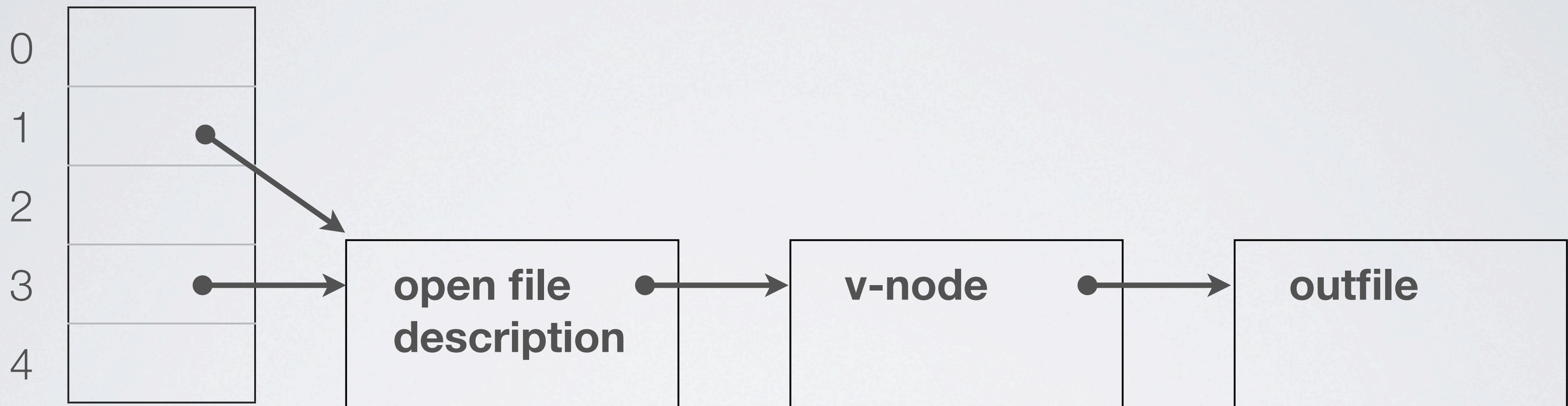



```
dup2(3, 1)
```

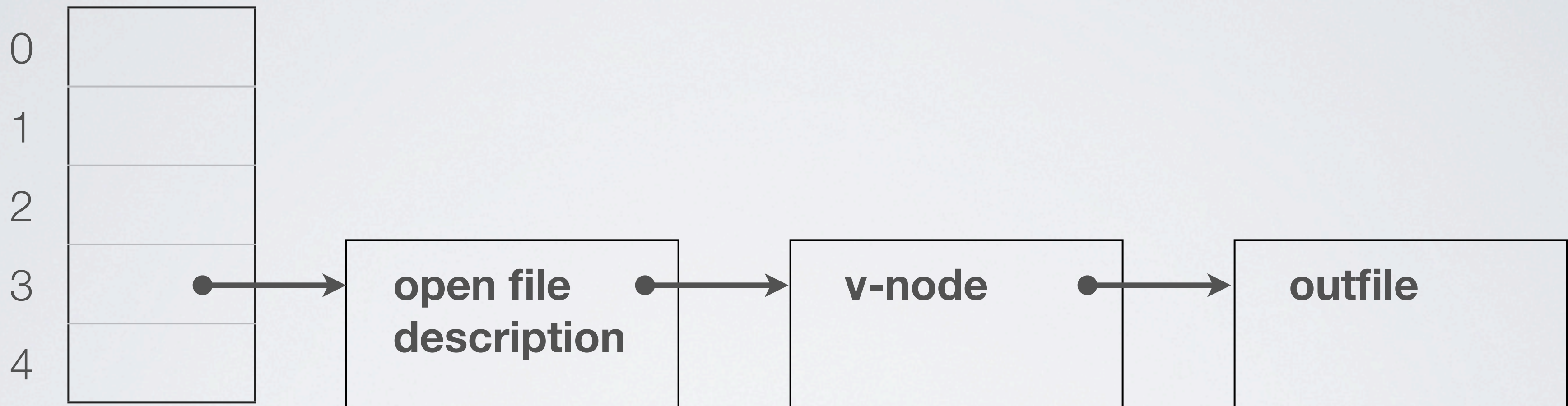


exec

```
int close (int fd);
```




```
close(1)
```




```
close(3)
```

open file description

v-node

outfile

I/O
read/write

```c
ssize_t read (int fd,
              void *buf,
              size_t nbyte);
```

short counts are possible

reasons:

- EOF
- out of space
- others?

```
ssize_t write (int fd,
               void *buf,
               size_t nbyte);
```

```
char buf[512];
int nread;
while ((nread = read(STDIN_FILENO, buf, 512)) > 0) {
    write(STDOUT_FILENO, buf, nread);
}
```

```c
off_t lseek (int fd,
             off_t offset,
             int whence);
```

seeking not possible
on character special files
(and some others)

back to those short counts...

(they’re a pain)

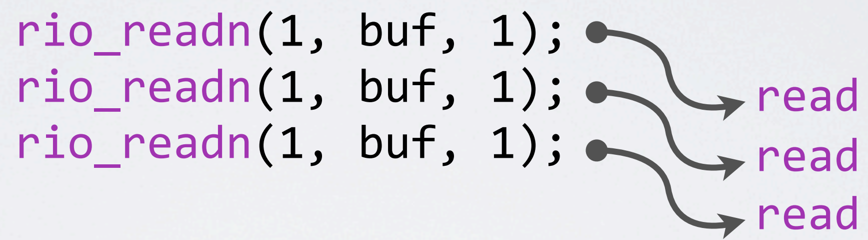
would like “**robust**” I/O

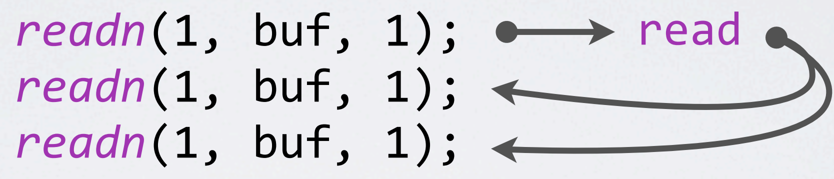
… implement at user level

```c
ssize_t rio_readn (int fd, void *usrbuf, size_t n)
{
    size_t nleft = n;
    ssize_t nread;
    char *bufp = usrbuf;

    while (nleft > 0) {
        if ((nread = read(fd, bufp, nleft)) < 0)
            return -1;
        else if (nread == 0)
            break;              /* EOF */
        nleft -= nread;
        bufp += nread;
    }
    return (n - nleft);         /* return >= 0 */
}
```

```
ssize_t rio_writen(int fd, void *usrbuf, size_t n)
{
    size_t nleft = n;
    ssize_t nwritten;
    char *bufp = usrbuf;

    while (nleft > 0) {
        if ((nwritten = write(fd, bufp, nleft)) <= 0)
            return -1;
        nleft -= nwritten;
        bufp += nwritten;
    }
    return n;
}
```

problem: traps are expensive!


rio_readn(1, buf, 1);
rio_readn(1, buf, 1);
rio_readn(1, buf, 1);
read
read
read


readn(1, buf, 1);
readn(1, buf, 1);
readn(1, buf, 1);
read

solution: buffering

user-space buffer

```c
typedef struct {
    int rio_fd;                /* descriptor for this internal buf */
    int rio_cnt;               /* unread bytes in internal buf */
    char *rio_bufptr;          /* next unread byte in internal buf */
    char rio_buf[RIO_BUFSIZE]; /* internal buffer */
} rio_t;
```

```
void rio_readinitb(rio_t *rp, int fd)
{
    rp->rio_fd = fd;
    rp->rio_cnt = 0;
    rp->rio_bufptr = rp->rio_buf;
}
```

```c
static ssize_t rio_read (rio_t *rp, char *usrbuf, size_t n)
{
    int cnt;

    while (rp->rio_cnt <= 0) {  /* refill if buf is empty */
        rp->rio_cnt = read(rp->rio_fd, rp->rio_buf, sizeof(rp->rio_buf));
        if (rp->rio_cnt < 0)
            return -1;
        else if (rp->rio_cnt == 0)  /* EOF */
            return 0;
        else
            rp->rio_bufptr = rp->rio_buf; /* reset buffer ptr */
    }

    /* Copy min(n, rp->rio_cnt) bytes from internal buf to user buf */
    cnt = n;
    if (rp->rio_cnt < n)
        cnt = rp->rio_cnt;
    memcpy(usrbuf, rp->rio_bufptr, cnt);
    rp->rio_bufptr += cnt;
    rp->rio_cnt -= cnt;

    return cnt;
}
```

```
ssize_t rio_readlineb (rio_t *rp, void *usrbuf, size_t maxlen)
{
    int n, rc;
    char c, *bufp = usrbuf;

    for (n = 1; n < maxlen; n++) {
        if ((rc = rio_read(rp, &c, 1)) == 1) {
            *bufp++ = c;
            if (c == '\n')
                break;
        } else if (rc == 0) {
            if (n == 1)
                return 0; /* EOF, no data read */
            else
                break;    /* EOF, some data was read */
        } else
            return -1;    /* error */
    }
    *bufp = 0;
    return n;
}
```

buffered output?

stdio.h

fclose fdopen feof ferror fflush
fgetc fgetln fgetpos fgets fgetwc
fgetws fopen fprintf fputc fputs
fread freopen fscanf fseek fsetpos
getc getchar gets mktemp perror
printf putc putchar puts remove
rewind scanf sprintf sscanf strerror
tmpfile ungetc vfprintf vprintf vscanf

note: **user-level** library

file “stream”

`FILE` struct

```
FILE *fopen(const char *path,
            const char *mode);
```

```c
FILE *fdopen(int fd, const char *mode);

int fclose(FILE *stream);

size_t fread(void *buf, size_t size, size_t nitems, FILE *stream);
size_t fwrite(void *buf, size_t size, size_t nitems, FILE *stream);

int fprintf(FILE *stream, const char *format, ...);
int fscanf(FILE *stream, const char *format, ...);
```

```c
#include <stdio.h>
extern FILE *stdin;  /* standard input (descriptor 0) */
extern FILE *stdout; /* standard output (descriptor 1) */
extern FILE *stderr; /* standard error (descriptor 2) */
```

```c
#include <stdio.h>

int main()
{
    printf("h");
    printf("e");
    printf("l");
    printf("l");
    printf("o");
    printf("\n");
    fflush(stdout);
    exit(0);
}
```

```
ada> strace ./hello
execve("./hello", ["./hello"], [/* 26 vars */]) = 0
...
open("/lib/libc.so.6", O_RDONLY)        = 3
...
write(1, "hello\n", 6...)               = 6
...
```

restrictions

output after input?
input after output?

output → input

fflush, fseek, fsetpos, rewind

input → output
fseek, fsetpos, rewind

no random access on character special files

```
int fd = open(...);

FILE *fpin  = fdopen(fd, “r”),
     *fpout = fdopen(fd, “w”);

/* use files */

fclose(fpin);
fclose(fpout);
```

Workaround

dangerous!