



CSE 333 Section AC

I/O, POSIX, and System Calls! (w/ Farrell & Travis)



Logistics

Due TODAY:

Homework 1 @ 9 pm

Due Monday:

Exercise 6 @ 11 am

POSIX

Posix is a family of standards specified by the IEEE. These standards maintains compatibility across variants of Unix-like operating systems by defining APIs and standards for basic I/O (file, terminal, and network) and for threading.

1) What does POSIX stand for?

Portable Operating System Interface

2) ` Why might a POSIX standard be beneficial? From an application perspective? Versus using the C stdio library?

- **More explicit control since read and write functions are system calls and you can directly access system resources.**
- **POSIX calls are unbuffered so you can implement your own buffer strategy on top of read()/write().**
- **There is no standard higher level API for network and other I/O devices**

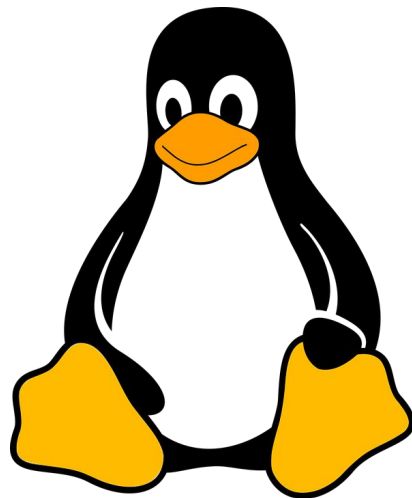
Review from Lecture

```
ssize_t read(int fd, void *buf, size_t count)
```

An error occurred	<code>result = -1</code> <code>errno = error</code>
Already at EOF	<code>result = 0</code>
Partial Read	<code>result < count</code>
Success!	<code>result == count</code>

New Scenario - Messy Roommate

- The Linux kernel is now your roommate
- There are N pieces of trash in the room
- There is a single trash can, `char bin[N]`
 - (For some reason, the trash goes in a particular order)
- You can tell your roommate to pick it up, but he/she is unreliable



New Scenario - Messy Roommate

NumTrash pickup(roomNum, trashCan, Amount)

<i>"I tried to start cleaning, but something came up"</i> (got hungry, had a midterm, room was locked, etc.)	NumTrash == -1 errno == excuse
<i>"You told me to pick up trash, but the room was already clean"</i>	NumTrash == 0
<i>"I picked up some of it, but then I got distracted by my favorite show on Netflix"</i>	NumTrash < Amount
<i>"I did it! I picked up all the trash!"</i>	NumTrash == Amount

```
NumTrash pickup(roomNum, trashCan, Amount)
```

How do we get the room clean?

- Use a loop. What's the (high level) goal?
 - Pick up all N pieces of trash
- What if the roommate returns -1 with an excuse?
 - If it's a valid excuse, stop telling them to pick up trash
 - If it's not, start over at the top of the loop
- What if the room is already clean?
 - Stop telling the roommate to pick up trash
- What if the roommate only picked up some of it?
 - Record how much they picked up, and tell them to pick up the rest
- What if the roommate picked up everything you asked?
 - Our goal has been reached!

```
NumTrash == -1, errno == excuse
```

```
NumTrash == 0
```

```
NumTrash < Amount
```

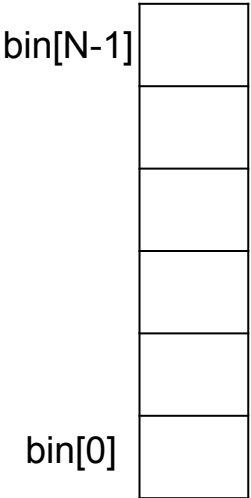
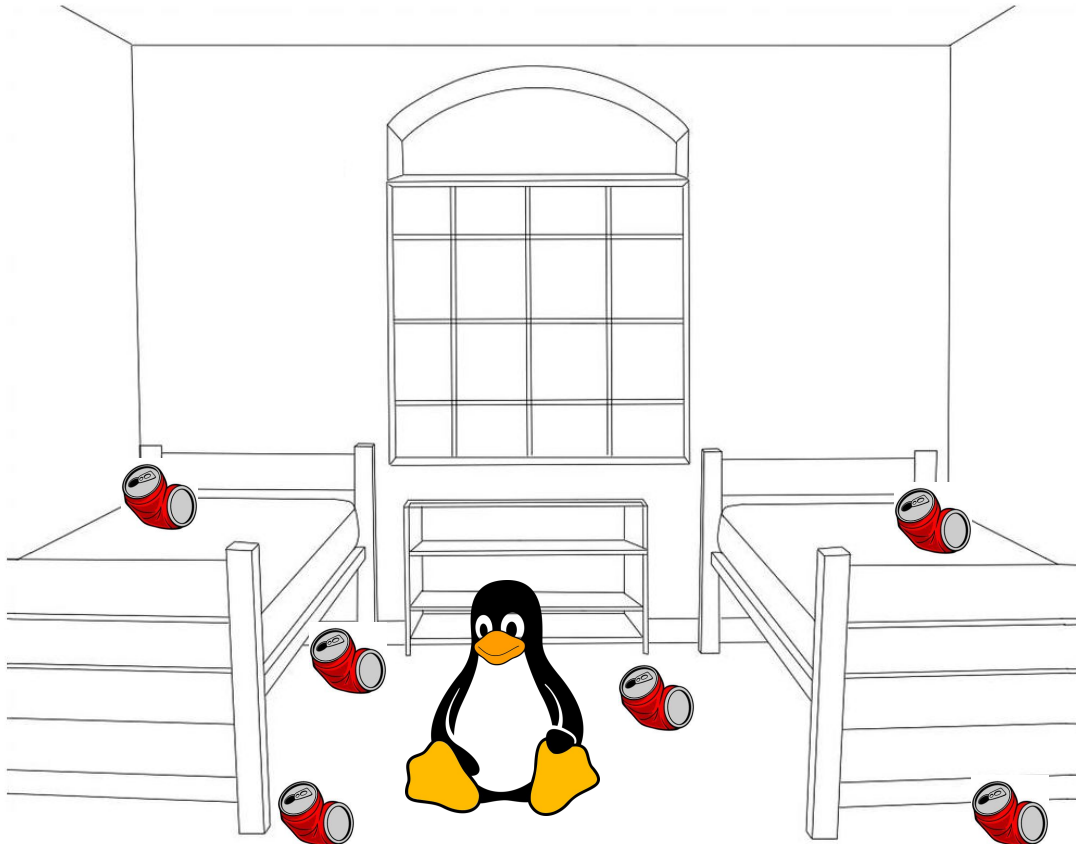
```
NumTrash == Amount
```

That's it!

NumTrash pickup(roomNum, trashCan, Amount)

How do we get the room clean?

NumTrash == -1, errno == excuse
NumTrash == 0
NumTrash < Amount
NumTrash == Amount



What do we do in the following scenarios?

NumTrash pickup(roomNum, trashCan, Amount)

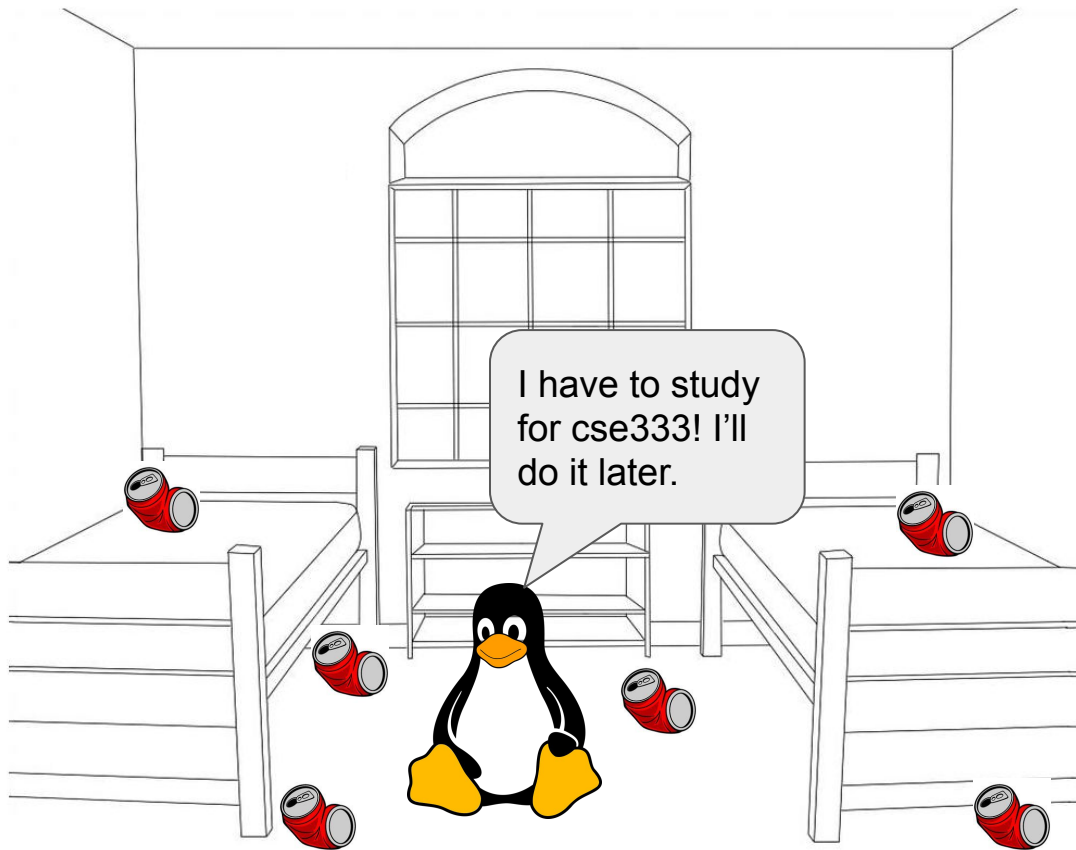
How do we get the room clean?

NumTrash == -1, errno == excuse

NumTrash == 0

NumTrash < Amount

NumTrash == Amount



bin[N-1]

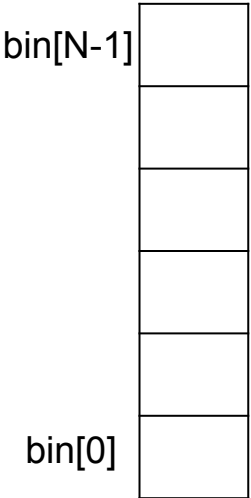
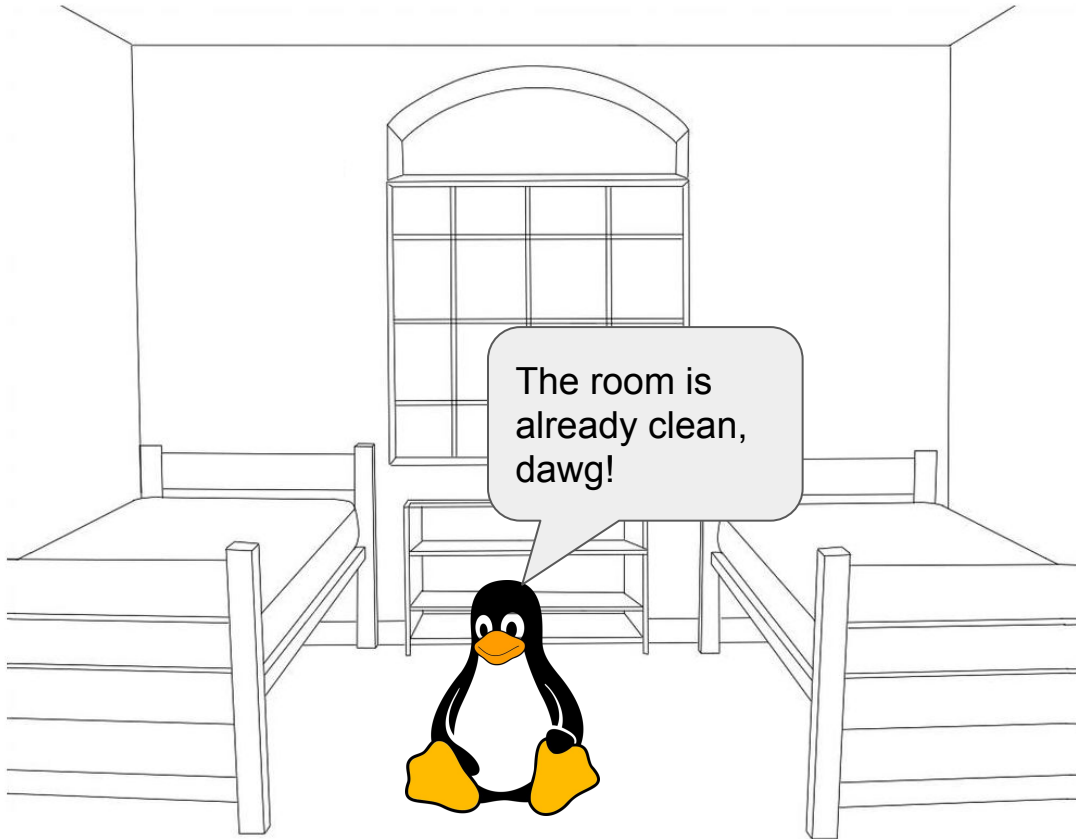
bin[0]

Decide if the excuse is reasonable, and either let it be or ask again.

NumTrash pickup(roomNum, trashCan, Amount)

How do we get the room clean?

NumTrash == -1, errno == excuse
NumTrash == 0
NumTrash < Amount
NumTrash == Amount

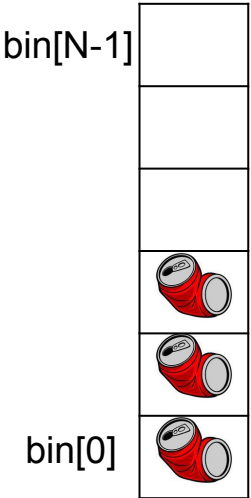
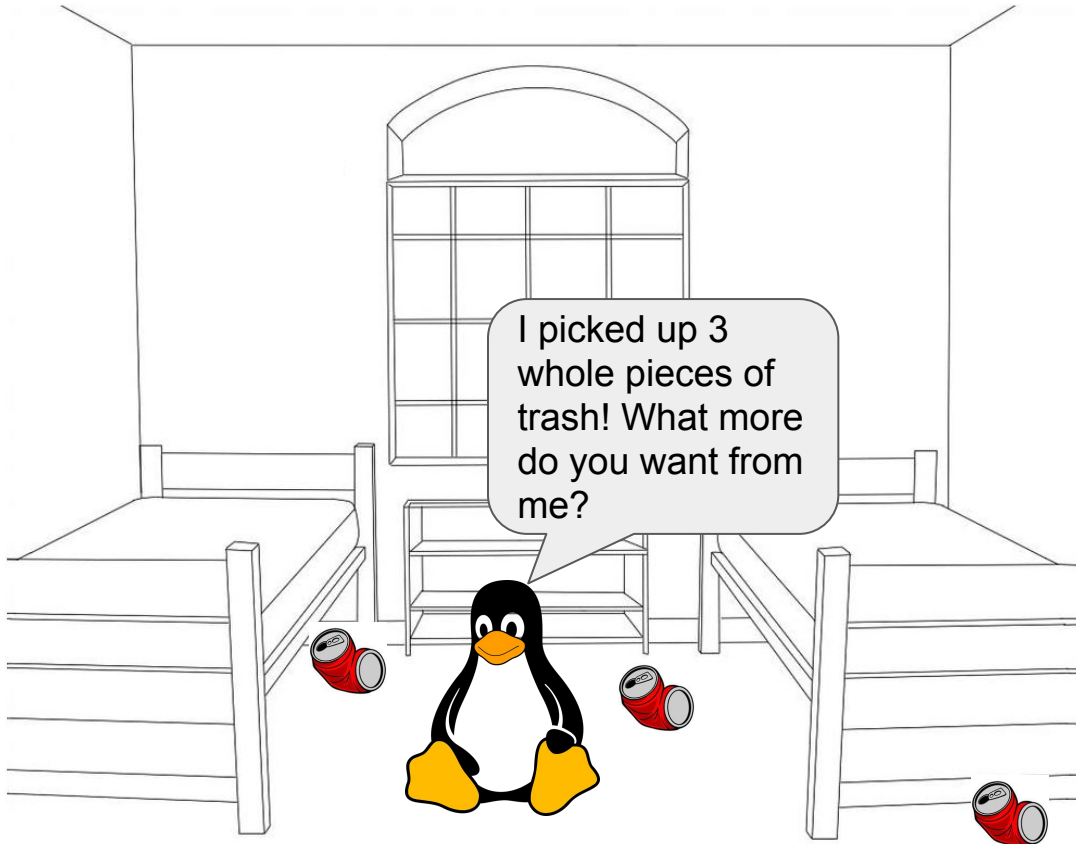


Stop asking them to clean the room!
There's nothing to do.

NumTrash pickup(roomNum, trashCan, Amount)

How do we get the room clean?

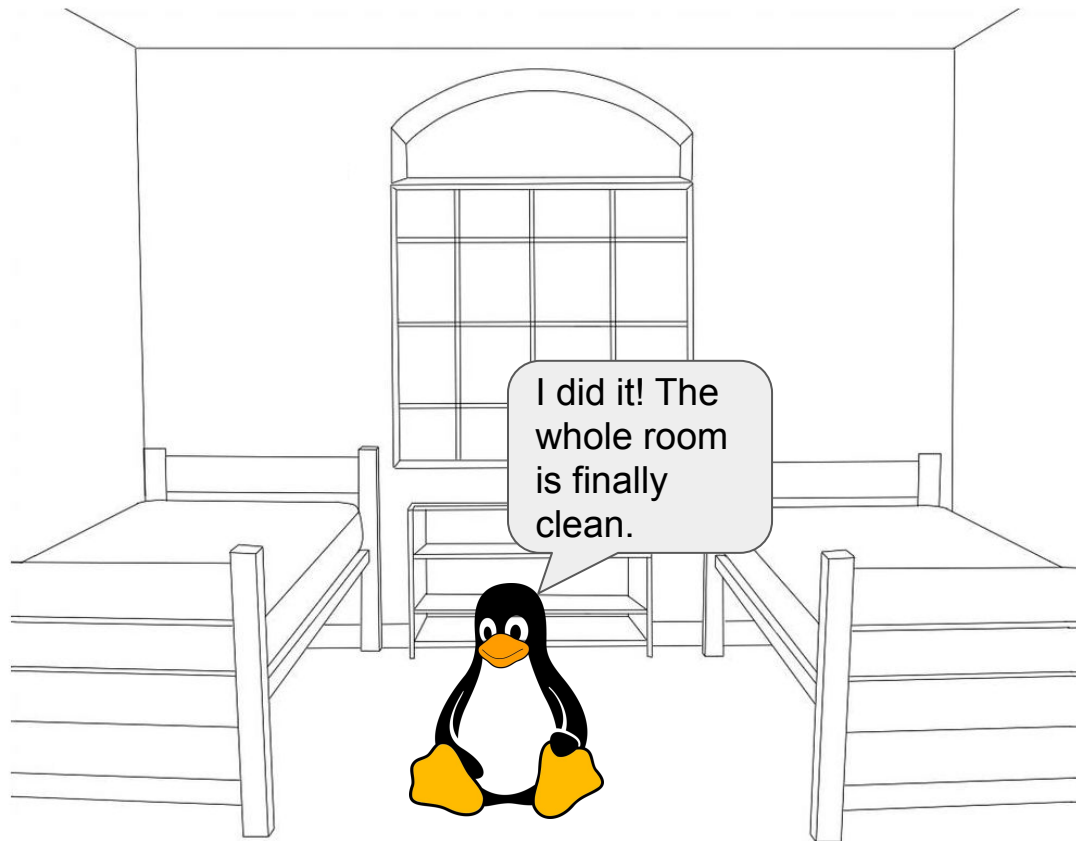
NumTrash == -1, errno == excuse
NumTrash == 0
NumTrash < Amount
NumTrash == Amount



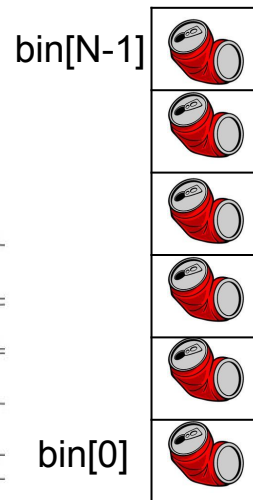
Ask them again to pick up the rest of it.

NumTrash pickup(roomNum, trashCan, Amount)

How do we get the room clean?



NumTrash == -1, errno == excuse
NumTrash == 0
NumTrash < Amount
NumTrash == Amount



They did what you asked, so stop asking them to pick up trash.

How do we get the room clean?

```
int pickedUp = 0;
while ( ----- ) {

}
}
```

NumTrash pickup(roomNum, trashCan, Amount)

NumTrash == -1, errno == excuse
NumTrash == 0
NumTrash < Amount
NumTrash == Amount

NumTrash pickup(roomNum, trashCan, Amount)

How do we get the room clean?

```
int pickedUp = 0;
while ( pickedUp < N ) {
    NumTrash = pickup( room, bin + pickedUp, N - pickedUp )
    if ( NumTrash == -1 ) {
        if ( excuse not reasonable )
            ask again
        stop asking and handle the excuse
    }
    if ( NumTrash == 0 ) // we over-estimated the trash
        stop asking since the room is clean
    add NumTrash to pickedUp
}
```

NumTrash == -1, errno == excuse
NumTrash == 0
NumTrash < Amount
NumTrash == Amount

NumTrash pickup(roomNum, trashCan, Amount)

How do we get the room clean?

```
int pickedUp = 0;
while ( pickedUp < N ) {
    result = pickup( room, bin + pickedUp, N - pickedUp )
    if ( result == -1 ) {
        if ( errno == E_BUSY_NETFLIX )
            continue;
        break;
    }
    if ( result == 0 )
        break;
    pickedUp += result;
}
```

NumTrash == -1, errno == excuse
NumTrash == 0
NumTrash < Amount
NumTrash == Amount

Some Final Notes...

We assumed that there were exactly N pieces of trash (N bytes of data that we wanted to read from a file). How can we modify our solution if we don't know N?

(Answer): Keep trying to read(. . .) until we get 0 back (EOF / clean room)

We determine N dynamically by tracking the number of bytes read until this point, and use `malloc` to allocate more space as we read.

There is no one true loop.

Tailor your POSIX loops to the specifics of what you need!



Back to the worksheet (Q3)



```

int fd = _____; // open 333.txt
int n = ....;
char *buf = ..... ; // Assume buf initialized with size n
int result;

_____ ; // initialize variable for loop

... // code that populates buf happens here

while ( _____ ) {

    result = write( _____, _____, _____ );

    if (result == -1) {
        if (errno != EINTR) {
            // a real error happened, return an error result
            _____ ; // cleanup
            perror("Write failed");
            return -1;
        }
        continue; // EINTR happened, so loop around and try again
    }
    _____ ; // update loop variable
}
_____ ; // cleanup

```

```
int fd = open("333.txt", O_WRONLY); // open 333.txt
int n = ....;
char *buf = ..... ; // Assume buf initialized with size n
int result;

char *ptr = buf; // initialize variable for loop

... // code that populates buf happens here

while (ptr < buf + n) {

    result = write(fd, ptr, buf + n - ptr);

    if (result == -1) {
        if (errno != EINTR) {
            // a real error happened, return an error result
            close(fd); // cleanup
            perror("Write failed");
            return -1;
        }
        continue; // EINTR happened, so loop around and try again
    }
    ptr += result; // update loop variable
}
close(fd); // cleanup
```

More Posix!

- 4) Why is it important to store the return value from the `write()` function? Why do we not check for a return value of 0 like we do for `read()`?
- 5) Why is it important to remember to call the `close()` function once you have finished working on a file?

More Posix!

- 4) Why is it important to store the return value from the `write()` function? Why do we not check for a return value of 0 like we do for `read()`?

**write() may not actually write all the bytes specified in count.
Writing adds length to your file, so you don't need to check for end of file.**

- 5) Why is it important to remember to call the `close()` function once you have finished working on a file?

More Posix!

- 4) Why is it important to store the return value from the `write()` function? Why do we not check for a return value of 0 like we do for `read()`?

**write() may not actually write all the bytes specified in count.
Writing adds length to your file, so you don't need to check for end of file.**

- 5) Why is it important to remember to call the `close()` function once you have finished working on a file?

In order to free resources i.e. other processes can acquire locks on those files.

Exercise:

- 6) Given the name of a file as a command-line argument, write a C program that is analogous to `cat`, *i.e.* one that prints the contents of the file to `stdout`. Handle any errors!
Example usage: `./filedump <path>` where `<path>` can be absolute or relative.

```
int main(int argc, char** argv) {
    /* 1. Check to make sure we have a valid command line arguments */

    /* 2. Open the file, use O_RDONLY flag */

    /* 3. Read from the file and write it to standard out. Try doing
       this without using printf() and instead have write() pipe to
       Stdout (take a look at STDOUT_FILENO). It might be helpful
       to initialize a buffer variable (of size 1024 bytes should be
       fine) to pass in to read() andwrite(). */

    /*4. Clean up */

}
```



```
int main(int argc, char** argv) {
    /* 1. Check to make sure we have a valid command line arguments */

    /* 2. Open the file, use O_RDONLY flag */

    /* 3. Read from the file and write it to standard out. Try doing
    this without using printf() and instead have write() pipe to
    Stdout (take a look at STDOUT_FILENO). It might be helpful
    to initialize a buffer variable (of size 1024 bytes should be
    fine) to pass in to read() and write(). */

    /*4. Clean up */
}
```

```
int main(int argc, char** argv){
    if (argc != 2) {
        fprintf(stderr, "Usage: ./filedump <filename>\n");
        exit(1);
    }
    int fd = open(argv[1], O_RDONLY);
    if (fd == -1) {
        fprintf(stderr, "Could not open file for reading\n");
        exit(1);
    }
    char buf[SIZE];
    ssize_t len;
    do {
        len = read(fd, buf, SIZE);
        if (len == -1) {
            if (errno != EINTR) {
                close(fd);
                perror(NULL);
                exit(1);
            }
            continue;
        }
        size_t total = 0;
        ssize_t wlen;
        while (total < len) {
            wlen = write(1, buf + total, len - total);
            if (wlen == -1) {
                if (errno != EINTR) {
                    close(fd);
                    perror(NULL);
                    exit(1);
                }
                continue;
            }
            total += wlen;
        }
    } while (len > 0);
    close(fd);
    return 0;
}
```