

Please **read this entire assignment, every word**, before you start working on the code. There might be some things in here that make it easier to complete.

This lab consists of one part. **This lab is October 18<sup>th</sup> by midnight.** Submit a single gzipped tar file to **TEACH**. The single gzipped tar file should contain the all source files (C source [`*.c` and `*.h`] and the `Makefile`). Submitting your solutions before October 18<sup>th</sup> will earn you a 10% bonus. **You must have a `Makefile` to build this assignment.**



This assignment is **NOT** done in the `xv6` environment. It is done on `os2`.

## Working with inodes – `mystat` (100 Points)

For this part of this assignment, you will write a C program that will **display the inode meta data for each file given on the command line**. You must call your source file `mystat.c` and your program will be called `mystat`. **This program will run directly on `os2`, not within `xv6`.**

An example of how my program displays the `inode` data is shown the Figure 4. You might also want to look at the output from the `stat` command (the command not a system function, `man` section 1). Though not as pretty (or in some cases as complete as the replacement you will write), it is the standard for showing `inode` information.

**Requirements** for your program are:

1. **The output of your program should look exactly like mine.**
2. Display the file type (regular, directory, symbolic link, ...) as a human readable string. If the file is a symbolic link, look 1 step further to find the name of the file to which the symbolic link points. If the file to which the link points does not exist, indicate that. See Figure 4.
3. Display the `inode` value.
4. Display the device id.
5. Display the mode as both its **octal** value and its symbolic representation. The symbolic representation will be the `rwX` string for user, group, and other. See Figure 4 or `'ls -l'` for how this should look.
6. Show the hard link count.
7. Show both the `uid` and `gid` for the file, as both the symbolic values (names) and numeric values. **This will be pretty darn easy if you read through the list of suggested function calls.** See Figure 4 for how this should look.
8. File size, in bytes.
9. Block count.
10. Show the 3 time values in **local time/date**. This will be pretty darn easy if you read through the list of suggested function calls. See Figure 4 for how these appear.

```

chaneyr@os2 # ls -ld F*
lrwxrwxrwx. 1 chaneyr upg9313  4 Sep 30 12:55 FUNNYbroken -> JUNK
drwxrwxr-x. 2 chaneyr upg9313  0 Sep 30 12:52 FUNNYdir
prw-rw-r--. 1 chaneyr upg9313  0 Sep 30 12:54 FUNNYpipe
-rw-rw-r--. 5 chaneyr upg9313 878 Sep 30 12:53 FUNNYregfile
srwxrwxr--. 1 chaneyr upg9313  0 Sep 30 13:05 FUNNYsocket
lrwxrwxrwx. 1 chaneyr upg9313 12 Sep 30 12:54 FUNNYSymlink -> FUNNYregfile

```

Figure 1: A sample of files to use for your testing. Found in ~chaneyr/Classes/cs444/Labs/Lab3

System and function calls that I believe you will find interesting include: `stat()` and `lstat()` (you really want to do “man 2 stat” and read that man entry closely, all of it [yes really, all of it]), `readlink()`, `memset()`, `getpwuid()`, `getgrgid()`, `strcat()`, `localtime()`, and `strftime()`. Notice that `ctime()` is NOT in that list and you don’t want to use it. Since you must be able to show the file type if a file is a symbolic link, I encourage you consider `lstat()` over `stat()`.

My implementation is about 280 lines long, but there is some dead code in my file. I have code commented out to support features not required for your assignment. There is no complex logic for this application, just a lot of long code showing values from the `struct stat` structure from `sys/stat.h`. Honestly, the longest portion of your code will likely be devoted to displaying the symbolic representation of the mode. Formatting these strings is a little *awkweird*. I suggest you create a function. Don’t worry about sticky bits or set uid/gid bits. Do you know what sticky bits are for or how they used to be used?

You must be able to show the following file types:

- regular file,
- directory,
- character device,
- block device,
- FIFO/pipe,
- socket, and
- symbolic link (both a good one and a broken one).

When formatting the human readable time for the local time, I’d suggest you consider this “%Y-%m-%d %H:%M:%S %z (%Z) %a”, but read through the format options on `strftime()`. The executable version of my code is in the directory. You are welcome to run it to see the output.

I have some examples of both a FIFO/pipe, a socket, symbolic link in my Lab3 directory for you to use in testing (the FUNNY\* files, Figure 2). You can

find a block device as

/dev/sda and a

character device as

/dev/sg0. See Figure 3.

```

chaneyr@flip2 # d /dev/sda /dev/sg0
brw-rw----. 1 root disk  8, 0 Oct  6 05:03 /dev/sda
crw-----. 1 root root 21, 0 Oct  6 04:59 /dev/sg0

```

Figure 3: Block and character files.

You must have a `Makefile` that builds the `mystat` program, **without any warnings from the compiler**. You must use the following `gcc` command line options (with `gcc` as your compiler) in your `Makefile` when compiling your code:

```
-g  
-Wall  
-Wshadow  
-Wunreachable-code  
-Wredundant-decls  
-Wmissing-declarations  
-Wold-style-definition  
-Wmissing-prototypes  
-Wdeclaration-after-statement
```

When we grade your program, we will issue the following commands to build your executable.

```
make clean  
make all
```

**There must be zero (0) warnings from the compiler. If your program compiles and produces warnings (using the above command line options for gcc), then it is a zero.**

## Final note

The labs in this course are intended to give you basic skills. In later labs, we will *assume* that you have mastered the skills introduced in earlier labs. **If you don't understand, ask questions.**

For those of you have read this far and are actually reading the entire assignment, thank you. As a reward, if you look at the section 2 man page for `stat()`, down near the bottom, I think you'll like it. That is:

```
man 2 stat
```

A broken symbolic link.

A directory.

A pipe.

A regular file.

A socket.

A symbolic link.

Figure 4: The `inode` information of the `FUNNY` files. Sorry the image is so small.