

Released: September 9th, 2022
Due: September 25, 2022, 11:55 PM EST
TA Live Q&A: September 10th, 2022, 2:00 PM EST (Daniel; [Recording Link](#))
Slack: [#homework-questions](#) (student discussions)
[#anubis-questions](#) (student discussions)
[#oncall-ta](#) (TA help)

CS-GY 6233 - Fall 2022

Homework #1

xv6 disk usage user program

-
- | | |
|--|----------|
| 1. Intro to xv6 and Anubis | [-- pts] |
| 2. Hello, World! | [20pts] |
| 3. Implementing <i>du</i> program | [50pts] |
| 4. Flags/Args for your <i>du</i> program | [30pts] |
-

ALL STUDENTS: AUTOGRADER INTEGRATION INSTRUCTIONS:

We will also have you add a user binary “casegen” (which we have written for you) for this assignment. This program sets up the xv6 filesystem for you (and us) so that we can test the *du* command without having timeouts using purely shell commands. You are free to use the program for your own testing as well (see the instructions inside for usage details).

We really appreciate everyone’s patience while we are getting everything setup.

Please go to [this GitHub gist](#), and paste “casegen.c” to the appropriate position within your xv6 repository. Thanks!

Alert:
Anubis outage occurred **Sept 13 8:15PM to 10:30PM**;
[See this slack announcement for details](#)

- For this assignment, you will work exclusively with the **xv6** operating system
- You will use a class-provided container, hosted on **Anubis**. This container:
 - Automatically manages a private GitHub repository for you, and autosaves your work
 - Already contains the xv6 source code and the necessary compiler/linker
 - Provides you with a VSCode web interface for development
 - Will 'autograde' certain parts of your code (visible case results only)
- For those of you with a C background, or those of you adept at Googling for C functions, it is important to note that xv6 is not compiled with the C standard library.
 - While a lot of the subparts of the assignment might look "easy," do not wait until the last minute to begin work.
 - There are many nuances to what functions and patterns will work within xv6, and a lot of these nuances are learned only through practice.
- The autograder is not guaranteed to be available (the TAs and the Anubis team will maintain a best-effort SLA).
 - Designing your own test cases is important.
 - Read the requirements carefully, ask questions, and prepare for somewhat-unexpected test cases.
 - **Your final submission will be graded against new/different cases than are present in the autograder.**

Where we're going with HW1...

In 1971, Dennis Ritchie released a tool named '[du](#)' (disk usage) for AT&T's Unix operating system, and throughout the years this utility has become quite ubiquitous across *nix OSs.

Disk Usage does just that; it calculates the amount of data associated with file and directory entries within the filesystem. Various options (called flags and/or arguments) can change this behavior – for this assignment, we will implement a generic 'du', as well as a very small subset of flags, for the xv6 OS.

Related Commands (already available in xv6): 'ls', 'mkdir', 'rm'

Part 1 - Boot xv6 from Anubis

1.A - Setting up Anubis

Whether working in pairs or not, it's important for each team member to perform the following:

1. You need a GitHub account for this course and to use Anubis. Create an account now if you do not have one already. It does not matter whether you use your NYU email for this account or not.
2. Visit anubis-lms.io and sign in with your NYU credentials.
3. Within Anubis, our first goal is to connect your GitHub account to the system. To do this, click the hamburger at the upper left to open the side panel, and select "Profile." Connect your GitHub account to Anubis.
4. Join our course by clicking on "Courses" on the left, and adding our course with code:
ilovegrados

1.B - Initializing Assignments

Again, each team member should perform the following:

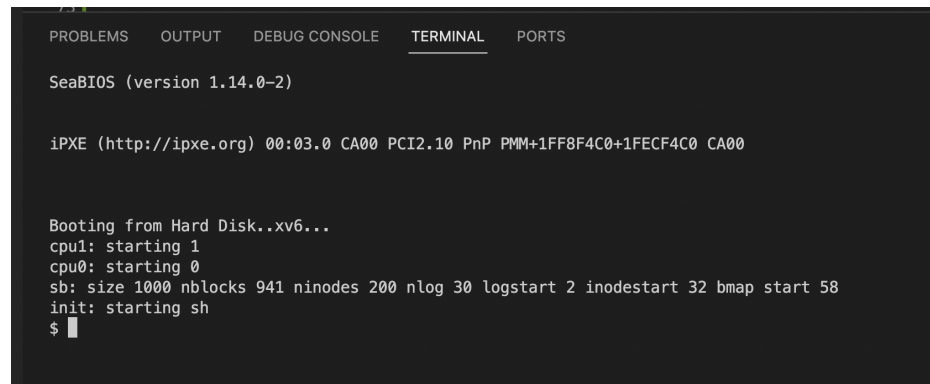
1. In the "Assignments" section, you should see Homework 1 listed. Open it.
2. Any time you start an assignment, your first step is to click on the "create a repo" button at the top right corner
 - a. Clicking on this button will create a private repository that contains the source code for the NYU version of xv6 (or other starter code if an assignment is outside of xv6)
 - b. The repository is **owned by Anubis** and shared with you as a **collaborator**
 - c. All of your work should be completed within this repository, and the only committer can be Anubis (through the cloud IDE).
 - d. **The state of this repository at the time of the due-date is what will be graded.**
3. Immediately after the repository has been created by Anubis, go and claim it! (Seriously... immediately... these invitations expire and they cannot be resent) – click "view repo" where the create repo button used to be, and you will be taken to an invitation page on GitHub.

1.C - First Boot

Start up your container and your IDE by clicking on "Open Anubis Cloud IDE" and follow the prompts. This will open up a VScode window in your browser. Click first on the folder icons at the top left to see the contents of your container (which should match your repository in github), and then click on Terminal on the menu bar in order to start a terminal on your Anubis container.

You should be able to boot xv6 immediately. Here are some commands for setting up, booting, and tearing down xv6 (in your vscode session on Anubis, open a terminal and execute these from the root directory of the assignment code where the Makefile is located):

1. `make qemu` will compile, link, prepare the filesystem, and boot xv6 using QEMU emulator.



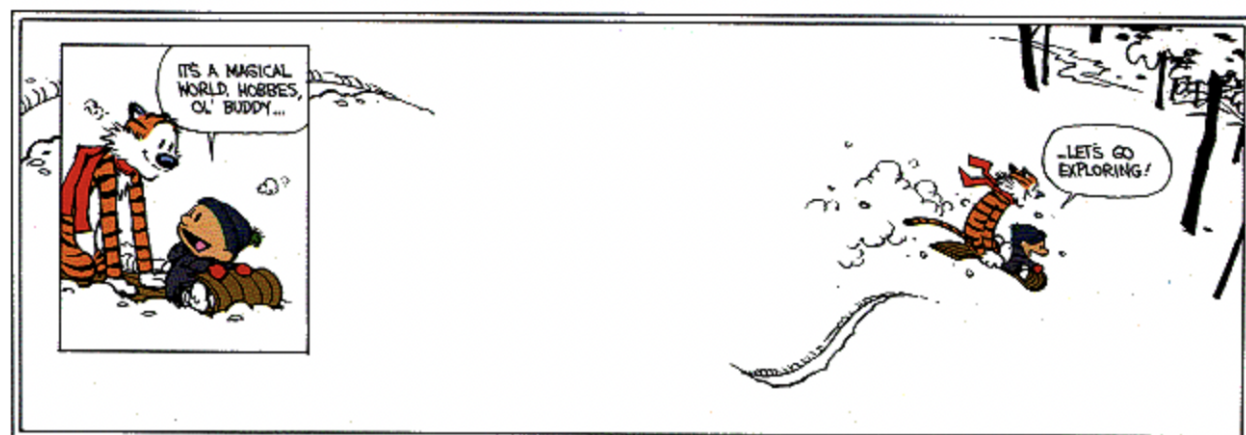
```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

SeaBIOS (version 1.14.0-2)

iPXE (http://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8F4C0+1FECF4C0 CA00

Booting from Hard Disk..xv6...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$
```

- a. You should see something similar. You now have a shell. Try something like `ls` to see which binaries are already built and available to you within user space.
2. In order to exit QEMU itself, the following key combination/sequence is required: `[CTRL+A]` then `[X]`. If you're on a Mac, sometimes this works, sometimes it doesn't – we're investigating. You can shut down xv6 by simply closing the terminal itself and opening a new one.
 3. This should return you back to your container's shell. You'll see that `make` has generated quite a few new files in the subdirectories. For this course, we're using "in-tree" builds, meaning your compiled artifacts will live in the same directory as your source code. **DON'T DELETE THINGS ON YOUR OWN.** Instead:
 4. `make clean` will remove all of the artifacts that were built, returning you to your original state of just the source files. Any changes to those source files remain, and any new source files you create will remain, it just simply removes any compiled artifacts that it generated.
 5. Helpful Hint
 - a. Within xv6 itself, there are some provided test binaries that can verify the integrity of the OS. `usertests` in particular is a great way to verify that all parts of the kernel are operating as expected. You won't need this for this assignment because you won't be modifying the kernel, but it's a helpful tool to have around when you do start making changes inside the kernel in later assignments. Feel free to run it now to see what all is tested (and be warned that it takes a few minutes to complete)



Part 2 - Hello, World!

Objective:

Create a user program named "hello" that prints "Hello, World!" to the console.

User call:

```
hello
```

Expected Output:

```
Hello, World!
```

(you must include a single newline character after the !)

IMPORTANT REQUIREMENT:

EVERY user binary in xv6 must use the xv6 system call `exit()` to return from the `main()` function. Some binaries also use `exit()` as a way of terminating when there are unhandled or invalid program states, but at the very least you must use `exit()` at the end of your `main()` (instead of returning an integer).

Other Suggestions:

- Review the source code for any of the existing user programs, and write a simple program that prints to the console.
- The output to the console must match the text in green exactly.
- In order to compile, link, and include your new program within the filesystem, find the entries in the `Makefile` that correspond to the program `ls`, and try to match them.
 - Hint: the xv6 compilation process runs a tool called `mkfs` that collects compiled binaries that begin with an underscore (like, `_hello`) and places them in the root directory of xv6.
 - Follow the pattern in the `Makefile` for how to correctly name your `*.c` file, and where to add the entry for it in `Makefile` so that `make` correctly completes this process.

Part 3 - Implementing *du* program

If you are familiar with this command already, please note that for Part 3 and Part 4 of this assignment, we are changing the default behavior of *du*¹ to better reflect points/difficulty.

Objective:

Create a user program named “*du*” that prints a list of the files in the current directory (where the program was run), with their on-disk size in bytes. The final line prints the sum of all of the sizes of the files within the directory.

User call:

```
du
```

(no arguments or flags)

Expected Output: (Correction on output – added “./” prefix)

```
34 ./file1
234 ./file2
1 ./file3
34577 ./file4
34846 .
```

(you must include a single newline character after your final line)

- Each line is formatted as “%d %s” – “{size in bytes}{space}{filename}”
- The final line is the sum of the sizes of all files in the directory where *du* was called.

Expected Behaviors:

- Size is considered to be the byte count stored in the xv6 `stat` structure.
- Only `T_FILE` types should be included in the output and in the calculation of the directory size.
- This implementation (with no arguments or flags) should not recurse into subdirectories.
- Your program should be able to be run from within any directory within xv6, and to report the files within that directory correctly.
- Hints:
 - Analyze the existing `ls` program to understand how its output relates to this new program’s requirements, as well as for ideas about how to structure your own program.

¹ Part 3 most similarly resembles the *nix command ``du -ab -d 1``, but we will not include directories in our version.

Part 4 - Flags/Args for your *du* program

Just as in regular C, the `main()` function for an xv6 user program can accept the following signature:

```
int main(int argc, char *argv[]){ ... }
```

The first variable indicates the number of space delimited arguments that were invoked by the user, and the second is an array of those arguments, already split by spaces.

Objective:

Add functionality to your prior *du* program so that it can handle various combinations of the following flags/arguments:

- `-k`
 - Report the number of *blocks* each entry is allocated in the filesystem (instead of bytes).
- `-t [threshold]`
 - Filter files based on a lower-bound threshold of `> [threshold]` bytes.
- `[file | directory]`
 - Run the command on a specific file, or on a different directory than where it was originally called.
- **10 points extra credit:**
 - `-r`
 - Recursively follow all subdirectories, outputting entries for each of their files, and including an output line for each subdirectory total.

User call examples:

```
du -k
```

```
du -t 1024
```

```
du subdirectory/
```

```
du file
```

```
du -t 1024 -k subdirectory/
```



```
du -k -t 100 -r subdirectory/  
credit}
```

{extra

Flag combinations (in any order):

-k	-t [threshold]	Expected Behavior
YES	NO	As described above, output the total blocks allocated; pay close attention to the order of operations of the sum of its blocks when outputting the summary line.
NO	YES	As described above, output only files that are larger than [threshold] bytes
YES	YES	Apply threshold first, and report on blocks for only those files that are above the threshold in bytes . The threshold should always be in bytes.

- Adding “[file]” at the end of any of the above flags will output an entry for just that file (and the summary line will match that file’s line).
- Adding “[directory]” at the end of any of the above will run the command from the specified directory as the starting path (and the summary line will output the sum for the directory).
- Extra Credit - adding “-r” to any of the above will recursively apply any other flags to files in each subdirectory. You may assume that the tested file structure will not have a tree depth more than 3 subdirectories (with the 4th level being leaf files).
- As a general rule for the course, if a flag is repeated, or an unrecognized flag/argument is included by the user, you should exit and output “check usage.\n”

Expected Output Examples: (Correction on output – added “./” prefix for implicit ‘.’)

```
du -k  
1 ./file1  
1 ./file2  
1 ./file3  
68 ./file4  
71 .
```

```
du -z  
check usage.
```

```
du -t 35  
234 ./file2  
34577 ./file4
```

```
34811 .
```

```
du -k -t 35
```

```
1 ./file2
68 ./file4
69 .
```

```
du input_file
```

```
235 input_file
235 input_file
```

```
du input_dir
```

```
546 input_dir/file8
23 input_dir/file9
68 input_dir/file10
637 input_dir
```

```
du -k -t 45 input_dir/
```

```
2 input_dir/file8
1 input_dir/file10
3 input_dir/
```

```
du -r input_dir
```

```
45 input_dir/subdir/file1
23 input_dir/subdir/file2
29 input_dir/subdir/file3
97 input_dir/subdir
45 input_dir/subdir2/file1
34 input_dir/subdir2/file2
79 input_dir/subdir2
176 input_dir
```

(in all cases, you must include a single newline character after your final line)

- Each line is formatted as "%d %s" – "{size in bytes or blocks}{space}{filename}"
- The final line is the sum of the sizes of all files in the directory, based on bytes or blocks.
- Use the xv6 block size that is already defined as a macro variable in `fs.h`.
- Hints:
 - Pay attention to directories named "." and ".." – these have special meanings, and the example outputs have been designed to help you discern when to output the summary line(s) as "." and when to output it as a name.
 - Refer to page 17-19 of the xv6 manual for information on the file system of xv6.

*Part 4 most similarly resembles the following *nix commands:*

1. `du -k` = ``du -a -d 1`` but we will not include directory entries.
2. `du -t 35` = ``du -ab -d 1 -t 35`` but we will not include directory entries.
3. `du -k -t 35` = ``du -a -d 1 -t 35`` but we will not include directory entries.
4. `du input_file` = ``du -b input_file`` but we include a duplicate line.
5. `du input_dir` = ``du -ab -d 1 input_dir`` but we will not include directory entries.
6. `du -k -t 45 input_dir/` = ``du -a -d 1 -t 45 input_dir/`` but we will not include directory entries.
7. `du -r input_dir/` = ``du -ab input_dir/`` but they include an additional 4 * 1024 bytes for each directory entry itself. We are **NOT** doing that for this homework, but it's worth noting that directory entries take up space on their own as well.

Rubric

Part	Description	Deduction
Part 2, 3, and 4	No exit()	-10pts for 'hello' -10pts for 'du'
Part 2, 3, and 4	Program does not compile/appear within xv6	-20pts for 'hello' -50pts for 'du'
Administration	Incorrect BrightSpace Submission (see below)	-20pts poorly formatted -100pts if we can't determine who agreed to work together
Visible Autograder	Not passing visible autograde tests (we will adjust as necessary if there is not enough time with the autograder available to you)	-5pts per case (10 cases to be passed)
Hidden Autograder	Not passing all hidden cases	-10pts total
Notes	Max deduction	-100pts

Submission

If working in pairs:

BOTH teammates need to submit a response to the homework that will be posted in BrightSpace. This should look like:

TEAM MEMBER 1: [nyu email] - USE THIS ANUBIS SUBMISSION TEAM MEMBER 2: [nyu email]

Again, if both team members do not submit these messages, we will not have confirmation that you worked together and agree to share the grade.

If working individually:

Please submit a response to the homework that will be posted in BrightSpace. This should look like:

TEAM MEMBER 1: [nyu email] - USE THIS ANUBIS SUBMISSION

Academic Honesty

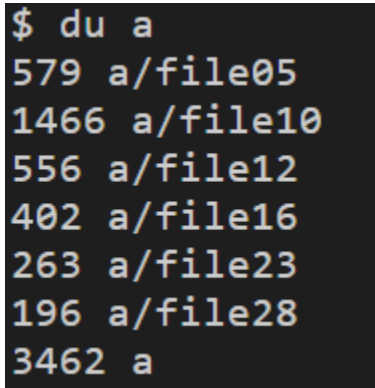
Aside from the narrow exception for collaboration on homework, all work submitted in this course must be your own. Cheating and plagiarism will not be tolerated. If you have any questions about a specific case, please ask the Prof/TAs. We will be checking for this!

NYU Tandon's Policy on Academic Misconduct:

<http://engineering.nyu.edu/academics/code-of-conduct/academic-misconduct>

Appendix - Casegen, Command Examples, and Expected Outputs

Please use the following example to visually confirm the output of your du.c program for a specified input. You will use the below instructions to generate a pre-populated file structure to test your program implementation on.

- Casegen Setup: After adding the casegen.c file to your user space and `$U/_casegen\` to the UPROGS list in the Makefile, you will be able to create and remove file structures in xv6 with the commands:
 - General Application:
 - `casegen setup [hexcode]`
 - `casegen teardown [hexcode]`
 - Specific Test: We will use the hexcode "a12f0" (0 is zero not capital o) for our testing of your programs implementation. Start by typing the casegen setup command below to create the file structure we will test with:
 - `casegen setup a12f0`
- Command Examples and Expected Outputs
 - Then try testing the following commands in xv6 and compare your results with the screenshots of the expected output:
 - Test 1:
 - Command: `du a`
 - Output:

```
$ du a
579 a/file05
1466 a/file10
556 a/file12
402 a/file16
263 a/file23
196 a/file28
3462 a
```
 - Test 2:
 - Command: `du -k a`
 - Output: (on next page)

```
$ du -k a
2 a/file05
3 a/file10
2 a/file12
1 a/file16
1 a/file23
1 a/file28
10 a
```

■ Test 3:

- Command: `du -t 560 a`
- Output:

```
$ du -t 560 a
579 a/file05
1466 a/file10
2045 a
```

■ Test 4:

- Command: `du -k -t 560 a`
- Output:

```
$ du -k -t 560 a
2 a/file05
3 a/file10
5 a
```

■ Test 5:

- Commands: (*Moves to "a" subfolder and runs du command*)
 - `cd a`
 - `../du .`
- Output: (*on next page*)

```
$ cd a
$ ../du .
579 ./file05
1466 ./file10
556 ./file12
402 ./file16
263 ./file23
196 ./file28
3462 .
$
```

■ Test 6:

- Commands: (*Moves to "a" subfolder and runs du command*)
 - `cd a`
 - `../du ./`
- Output:

```
$ cd a
$ ../du ./
579 ./file05
1466 ./file10
556 ./file12
402 ./file16
263 ./file23
196 ./file28
3462 ./
$
```

■ Test 7:

- Commands: (*Moves to "a" subfolder and runs du command*)
 - `cd a`
 - `/du`
- Output: (*on next page*)

```
$ cd a
$ /du
579 ./file05
1466 ./file10
556 ./file12
402 ./file16
263 ./file23
196 ./file28
3462 .
$
```

- Casegen Teardown: Finally, `cd` back to the root directory and run the casegen teardown command below to remove the file structure you created with the prior setup command:

- `casegen teardown a12f0`