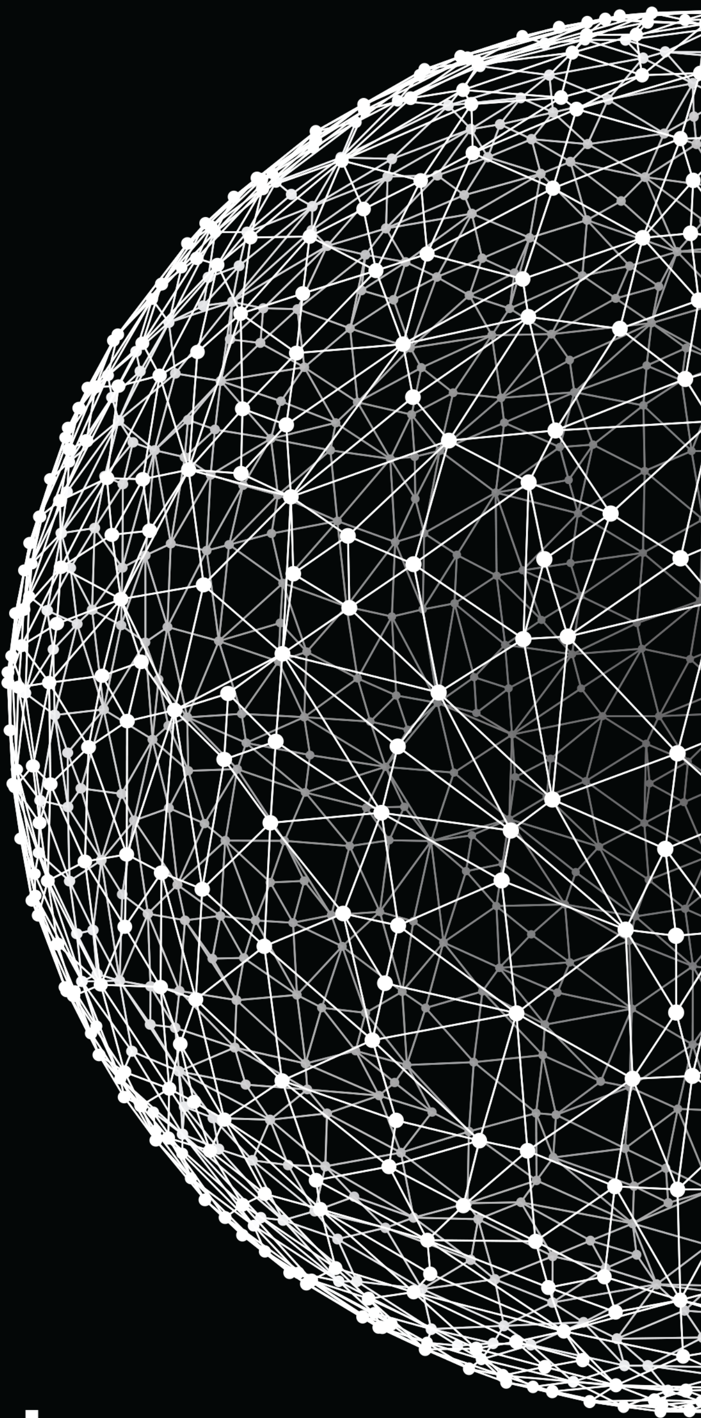
uls

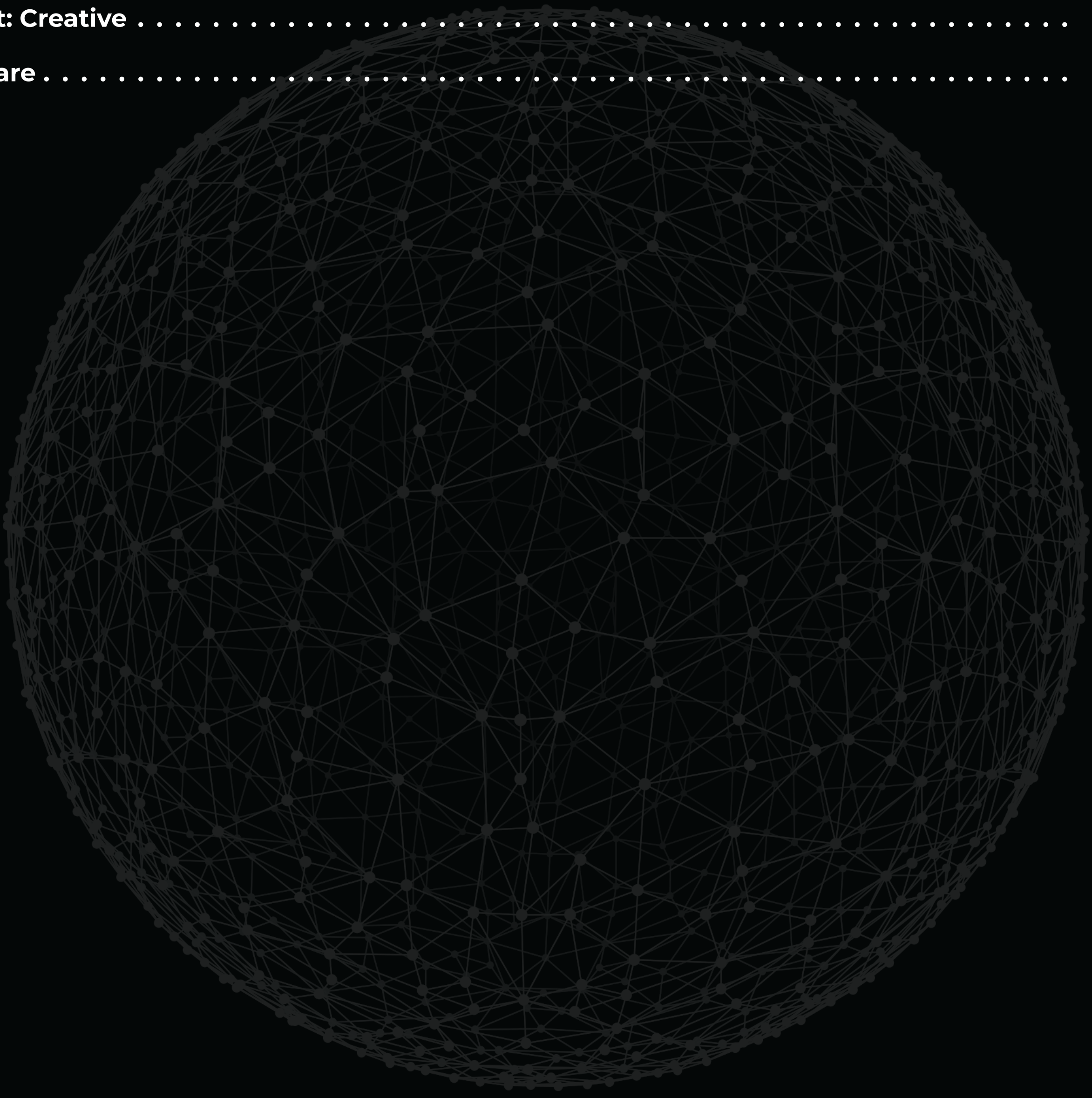
Track C

September 4, 2020



ucode connect

Contents



ucode connect

Engage

DESCRIPTION

Yooo!

The `ls` command is one of the most important commands when working in shell. It's time to find out how it works under the hood.

This challenge is designed to implement your own `uls`. A deep understanding of this utility will help you learn how a file system works in an operating system. Also, you'll learn how to programmatically interact with a file system using C.

Good luck, The One!

BIG IDEA

Data storage, persistence and management.

ESSENTIAL QUESTION

How does an OS interact with file systems?

CHALLENGE

Recode the system utility `ls`.

Investigate

GUIDING QUESTIONS

We invite you to find answers to the following questions. By researching and answering them, you will gain the knowledge necessary to complete the challenge. To find answers, ask the students around you and search the internet. We encourage you to ask as many questions as possible. Note down your findings and discuss them with your peers.

- What is a file?
- What is a directory?
- What are device files?
- Why do we need symbolic links?
- What is a file system?
- What is the UNIX file system?
- Are you familiar with the usual layout of directories in MacOS?
- What are command-line flags (options) and why are they useful?
- What flags of the ls command have you used?
- Why does a file system require file access control lists (ACL)?
- What is the role of extended file attributes in a file system?

GUIDING ACTIVITIES

Complete the following activities. Don't forget that you have a limited time to overcome the challenge. Use it wisely. Distribute tasks correctly.

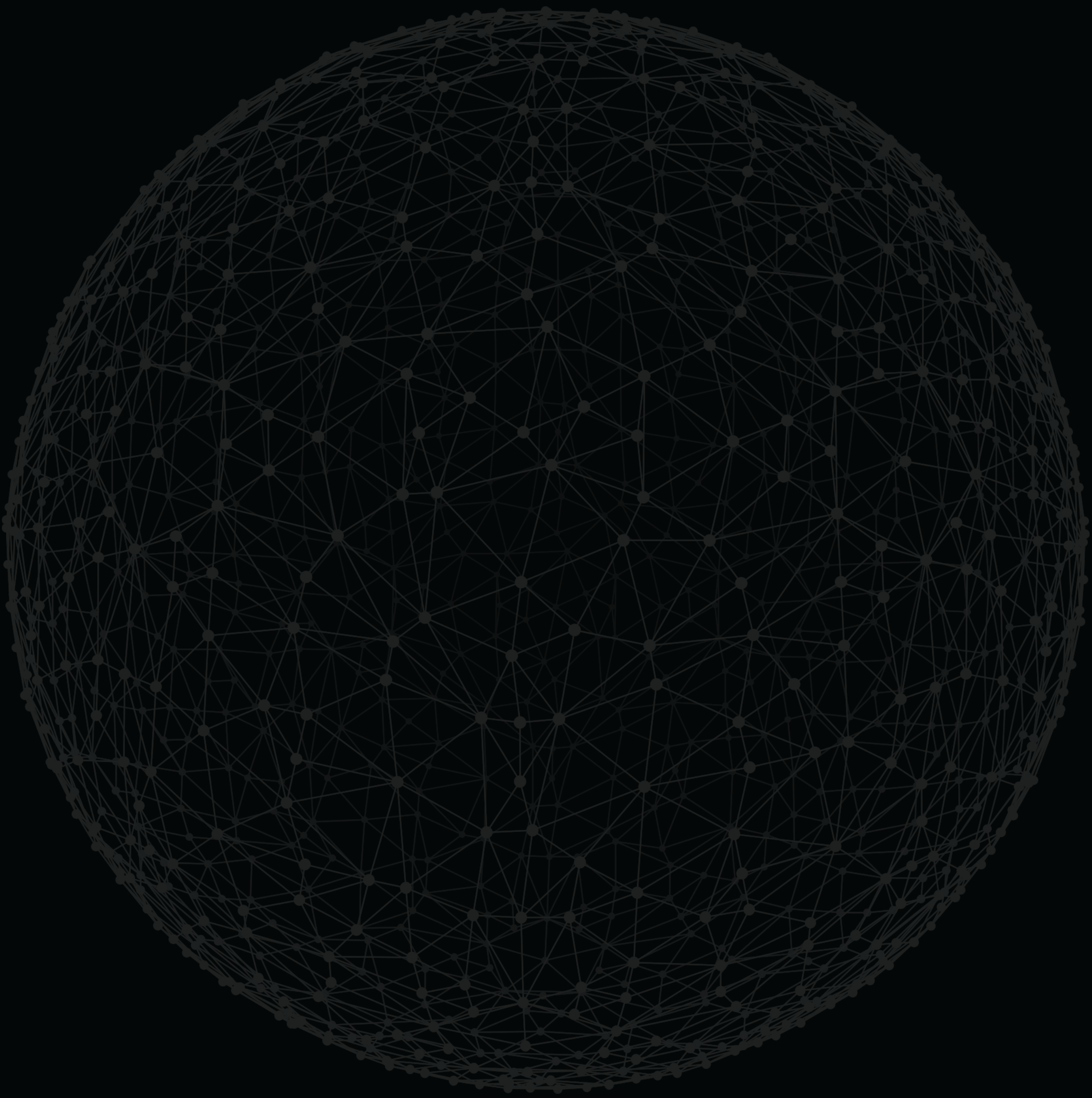
- Read the story carefully. Find out what you have to do.
- Read the man and try to understand how the standard utility `ls` works. This is your main guideline for this task.
- Experiment with the standard `ls`. Find out its features. This way, you'll find out additional information that is not listed in the man.
- Read the documentation for each solution component.
- Collaborate with other students to get a deeper understanding of the challenge. Learn from others and share your knowledge.
- Brainstorm with other students and find more than one solution.
- Clone your git repository that is issued on the challenge page.
- Distribute tasks among team members.
- Start developing your program. Try to implement your thoughts in code.
- Open Auditor and follow the rules.
- Check the work and check whether it works in the same way as the standard utility `ls`.
- Test your code.

ANALYSIS

Analyze your findings. What conclusions have you made after completing guiding questions and activities? In addition to your thoughts and conclusions, here are some more analysis results.

- Challenge has to be carried out by an entire team.
- Each team member must understand the challenge and realization, and be able to reproduce it individually.
- It is your responsibility to assemble the whole team. Phone calls, SMS, messengers are good ways to stay in touch.
- Be attentive to all statements of the story.
- The challenge must be performed in `C`.
- Perform only those tasks that are given in the story.
- The challenge must have the following structure:
  - `src` directory contains files with extension `.c`
  - `obj` directory contains files with extension `.o` (you must not push this directory in your repository, only `Makefile` creates it during compilation)
  - `inc` directory contains header files `.h`
  - `libmx` directory contains source files of your library including its `Makefile`. You must use it
  - `Makefile` that compiles the library `libmx` firstly and then compiles and builds `uls`
- You can proceed to `Act: Creative` only after you have completed all requirements in `Act: Basic`. But before you begin to complete the challenge, pay attention to the program's architecture. Take into account the fact that many features indicated in the `Act: Creative` require special architecture. And in order not to rewrite all the code somewhen, we recommend you initially determine what exactly you will do in the future. Note that the `Act: Basic` part gives the minimum points to validate the challenge.
- Complete the challenge according to the rules specified in the `Auditor`.
- Submit your files using the layout described in the story. Only useful files allowed, garbage shall not pass!
- Compile C-files with clang compiler and use these flags: `clang -std=c11 -Wall -Wextra -Werror -Wpedantic`.
- Pay attention to what is allowed. Use of forbidden stuff is considered a cheat and your challenge will be failed.
- Your program must manage memory allocations correctly. A memory that is no longer needed must be freed, otherwise, the challenge is considered incomplete.
- The solution will be checked and graded by students like you. `Peer-to-Peer learning`.
- Also, the challenge will pass automatic evaluation which is called `Oracle`.

- If you have any questions or don't understand something, ask other students or just Google it.



Act: Basic

ALLOWED

`write, malloc, malloc_size, free, perror, strerror, exit, opendir, readdir, closedir, stat, lstat, getpwuid, getgrgid, listxattr, getxattr, time, ctime, readlink, ioctl, isatty, acl_get_file, acl_to_text, acl_free, errno`

BINARY

`uls`

DESCRIPTION

Create a program that works as a system utility `ls`. You must implement:

- the usage: `usage: uls [-l] [file ...]`
- basic functionality of this command - list of directory contents without flags
- processing of `file` operands for files and directories
- the `-l` flag, which is one of the most useful flags
- the view of extended file attributes and access control lists (ACL)
- error handling, as in the original `ls`. Output `uls` as the program name instead of `ls` where necessary
- the multicolumn output format when the option `-l` isn't specified

CONSOLE OUTPUT

```
>./uls -z | cat -e
uls: illegal option -- z
usage: uls [-l] [file ...]
>./uls xxx
uls: xxx: No such file or directory
>echo $?
1
>./uls
dir1  dir2  dir3  file1 file2 file3 file4 file5
>./uls | cat -e
dir1$
dir2$
dir3$
file1$
file2$
file3$
file4$
file5$
>./uls dir1 dir2 file1 file2 file3
file1 file2 file3

dir1:
A.txt  E.txt  I.txt  M.txt  Q.txt  U.txt  Y.txt
B.txt  F.txt  J.txt  N.txt  R.txt  V.txt  Z.txt
C.txt  G.txt  K.txt  O.txt  S.txt  W.txt
D.txt  H.txt  L.txt  P.txt  T.txt  X.txt

dir2:    # empty directory
>./uls -l | cat -e
total 0
drwxr-xr-x  4 neo  staff  128 May 17 15:15 dir1$
drwxr-xr-x  2 neo  staff   64 May 17 14:57 dir2$
drwxr-xr-x  2 neo  staff   64 May 17 14:57 dir3$
-rw-r--r--  1 neo  staff    0 May 17 14:56 file1$
-rw-r--r--  1 neo  staff    0 May 17 14:56 file2$
-rw-r--r--  1 neo  staff    0 May 17 14:56 file3$
-rw-r--r--  1 neo  staff    0 May 17 14:56 file4$
-rw-r--r--  1 neo  staff    0 May 17 14:56 file5$
>
...    # find out more examples by yourself
```

FOLLOW THE WHITE RABBIT

```
man ls
man 4 tty
```

Act: Creative

DESCRIPTION

It is the place where your imagination and creativity plays a major role. Implement additional features to make the program better and more unique. Listed below are a few ideas you can add to your program. You can come up with everything you want to improve your program. Creative features:

- `-R` - if you want to implement this flag, but don't take it into account from the very beginning of the development process, you might have to rewrite your entire program. We highly recommend to implement this flag, because it will boost your brain, and, also, it is worth a lot of points in the evaluation.
- `-a` , `-A` - interesting flags for implementation both individually and in combination with the `-R` flag. Be careful, as there are some tricky parts to these flags
- `-G` - make your program colorful and fancy
- `-h` , `-@` , `-e` , `-T` , etc. - flags that are used in combination with other flags, for example with `-l`
- `-1` , `-C` , etc. - some simple flags. Note that this particular creative feature provides less additional points, than the other 2, since these 5 flags are rather simple to implement. may not be enough to validate the challenge with a positive mark
- `-r` , `-t` , `-u` , `-c` , `-S` , etc. - various sorting flags. All of them are interesting, but not all give a lot of points. Investigate their behavior
- read the man for additional cool flags

Of course, you must choose from those flags that are present in the original utility. Do not forget to add all implemented flags to the output about usage, as in the original `ls` utility.

Remember to optimize your algorithms! Compare the speed of your program with the original. Think about improving your program to speed it up.

Please do not consider adding other advanced features except those flags that the original `ls` provides. That is not the purpose of this challenge.

Share

PUBLISHING

Last but not least, the final stage of your work is to publish it. This allows you to share your challenges, solutions, and reflections with local and global audiences. During this stage, you will discover ways of getting external evaluation and feedback on your work. As a result, you will get the most out of the challenge, and get a better understanding of both your achievements and missteps.

To share your work, you can create:

- a text post, as a summary of your reflection
- charts, infographics or other ways to visualize your information
- a video, either of your work, or a reflection video
- an audio podcast. Record a story about your experience
- a photo report with a small post

Helpful tools:

- Canva - a good way to visualize your data
- QuickTime - an easy way to capture your screen, record video or audio

Examples of ways to share your experience:

- Facebook - create and share a post that will inspire your friends
- YouTube - upload an exciting video
- GitHub - share and describe your solution
- Telegraph - create a post that you can easily share on Telegram
- Instagram - share photos and stories from ucode. Don't forget to tag us :)

Share what you've learned and accomplished with your local community and the world. Use #ucode and #CBLWorld on social media.