Question 1:

Expanding on the implementation of the `ls` command and the `redirection operator` shown in class, using system calls, you need to implement the long form of the `ls` command (`ls -l`) on a specified directory and redirect the output to a text file (`list.txt`).

Explanation:

For example, your **.c** executable name is `myls` and you give a command line argument as the directory you want to perform the `ls` on. Your prompt will look like this.

```
./myls /DSTN/TestFolder
```

This translates to `ls -l /DSTN/TestFolder > list.txt`

If no command line argument is given (`./myls`), do the ls command on the current directory.

```
ls -l . > list.txt
```

Hint:

- Important system calls: `opendir(), readdir(), stat(), dup2(), exec family` system calls. You should use more system calls as required.
- Implement the ls command first. Then use that executable as the parameter for the exec call to implement the redirection operation.

Question 2:

- Write a script that can monitor the file system behavior of an application.
  - Select any application of your choice that interacts with the file system regularly (i.e. the application creates files, reads them etc.). It will be wise for you to not select a very complex application.
  - Use the `strace` tool to trace the system calls issued while running the application and log those in an output file. You can only trace the system calls that are taught in class. (Hint: use the `-e` and `-o` flags. Use more flags as required).
  - Write a **monitor (**in any language**)** that parses those system calls and tells the user in plain English language what is being done on the file system. (Hint: You will need to keep a track of the mapping of the fids and the filename - similar to the open file table)

    For example:
    - `Created directory <directory name>`
    - `Created file <filename>`
    - `Read from file <filename>`
    - `Written into file <filename>`
    - `Renamed <filename> to <filename2>`
    - `Deleted the file`
    - `Created a hard link <link name> for <filename>`
    - `Deleted directory <directory name>`

Question 3: (RAID Simulator with README)

- Run `./raid.py -h`
  - Familiarize with the options for the RAID simulator.
- In its basic mode, you can use it to understand how the different RAID levels map logical blocks to underlying disks and offsets.
  - Run `./raid.py -n 5 -L 0 -R 20`
  - We simulate five requests (-n 5), specifying RAID level zero (-L 0), and restrict the range of random requests to just the first twenty blocks of the RAID (-R 20). The result is a series of random reads to the first twenty blocks of the RAID; the simulator then asks you to guess which underlying disks/offsets were accessed to service the request, for each logical read.
    - `disk   = address % number_of_disks`
    - `offset = address / number_of_disks`
  - You can see the answers (once you've computed them!), by using the `-c` flag to compute the results.
    - `./raid.py -R 20 -n 5 -L 0 -c`
- You can also do this problem in reverse, with the `-r` flag.
- To get more variety, a different random seed (`-s`) can be given.
- You can also explore how writes behave (instead of just reads) with the `-w` flag, which specifies the "write fraction" of a workload, i.e., the fraction of requests that are writes. By default, it is set to zero, and thus the examples so far were 100% reads.
  - `./raid.py -R 20 -n 5 -L 1 -w 100 -c`
  - With writes, instead of generating just a single low-level disk operation, the RAID must of course update both disks, and hence two writes are issued.
- Other options:
  - The `-C` flag allows you to set the chunk size of the RAID, instead of using the default size of one 4-KB block per chunk. The size of each request can be similarly adjusted with the `-S` flag. The default workload accesses random blocks; use `-W` sequential to explore the behavior of sequential accesses. With RAID-5, two different layout schemes are available, left-symmetric and left-asymmetric; use `-5 LS` or `-5 LA` to try those out with RAID-5 (`-L 5`).
  - In timing mode (`-t`), the simulator uses an incredibly simple disk model to estimate how long a set of requests takes, instead of just focusing on mappings. In this mode, a random request takes 10 milliseconds, whereas a sequential request takes 0.1 milliseconds. The disk is assumed to have a tiny number of blocks per track (100), and a similarly small number of tracks (100). You can thus use the simulator to estimate RAID performance under some different workloads.

- **QUESTION:**
  a. How does the performance of each RAID level scale as the number of disks increases?
  b. Use the timing mode of the simulator to estimate the performance of 100 random reads to the RAID, while varying the RAID levels, using 4 disks.
  c. Run the timing mode with a sequential workload (`-W sequential`). How does the performance vary with RAID level, and when doing reads versus writes? What size should you write to a RAID when using RAID-4 or RAID-5?