# The problems of the lab

## Problem Set

1. Implement an application based on the example from `http://www.ciprianpungila.com/uvt/so2011/lab2/`, which modifies the behavior of the `-a`, `-b`, and `-c` options such that each of them can compute option arguments. The program should calculate the sum of all option arguments. For example:

   `<path_to_exec>/a.out -a 12 -b 24 -c 12`

   Should print:

   `48 (12 + 24 + 12)`

   Add a fourth (`-d`) and a fifth (`-e`) option based on the code found in Chapter 6 from `http://ciprianpungila.com/uvt/so2016/lab2/`.

   - When `-d` is enabled, the program computes the average of the first 100 prime numbers and prints the result.
   - The `-e` option requires an argument (`-e evalue`), and computes the average between the average of the first `cvalue` prime numbers and the average of the first `evalue` prime numbers.

   Example output for:

   `<path_to_exec>/a.out -a -b -c5 -d -e 8`

   `aflag = 1, bflag = 1, cvalue = 5, avg of first 100 primes = <value>, evalue = 8, AVG(5.6, 9.625) = 7.6125`

2. Build an application that recursively traverses a directory passed as a parameter and displays the size of all regular files in this directory.

3. Write a C program for Unix, callable from the command line as:

   `copytree path1 path2`

   This program copies the directory `path1`, including all its subdirectories, to the destination `path2`. If `path2` already exists, an error message is generated. If symbolic links are found, symbolic links will be created at the destination referring to the same file.

4. Build an application that recursively traverses a directory (specified as the first argument) and saves the sizes of all regular files into another file (specified as the second argument).

5. Implement a basic `cp`-like command that copies the contents of one file to another. Files are specified as arguments to the program (only 2 arguments are allowed).

6. Implement a basic `mv`-like command that moves the contents of one file to another. Files are specified as arguments to the program (only 2 arguments are allowed).

7. Explain the effect of the following code:

```
int i;
for (i = 1; i <= 10; i++)
    fork();
```

8. Write a C program for UNIX that creates 20 processes (including the parent). Each process prints 10 lines containing its type and PID. Afterward, the child processes terminate with different values, and the parent prints the values returned by the children.

9. Write a C application to recreate the same process hierarchy as depicted in a whiteboard illustration. Each process will print its PID and the parent's PID.

10. Write a C application that takes an integer argument (`N1`) from the command line, forks two child processes, and computes:

    (a) The sum of integers up to `N1` (first child).

    (b) The factorial of `N1` (second child).

    The parent waits for both children to finish, then prints "Done".

11. Write a C program to recreate the above program, but each child spawns subchildren that exit with respective index values. The parent captures the children's results, which include sums and products of subchildren's exit values.

12. Write a C application that reads a file, reverses its content, and uses a pipe for parent-child communication.

13. Extend the above application so the child sends back processed data:

    - `<int N><N chars><int M><M chars><int L><L chars><K chars>`
    - `N`: Number of digits, followed by the digits.
    - `M`: Number of uppercase letters, followed by the letters.
    - `L`: Number of lowercase letters, followed by the letters.
    - `K`: Remaining characters.

14. Write a C application that spawns two child processes, using pipes for parent-child communication. Process the file content as specified in the previous problem.

15. Implement a C application with a parent process sending `SIGUSR1` to a child. The child computes values using `a[n+1] = a[n] + 1/r` and prints `a[n]` for two cycles before exiting. The parent alternates between printing `*****`, `+++++`, and `-----`.

16. Write a C program using shared memory or semaphores to simulate a barber shop with 5 waiting chairs and customer processes.

17. Write a C++ program that creates three child processes:

    - Child 1: Sends its PID to Child 2 using a pipe.
    - Child 2: Reads Child 1's PID and prints it.
    - Child 3: Prints its parent's PID.

18. Write a C program that spawns 6 threads to find prime numbers between 2 and 10,000,000 that end in 7 or 9, limiting concurrency to 3 threads at a time using a semaphore.