1. Implement a shell -- multiple programming languages: C, Rust
    a. Create new processes, and execute them given commandline
    b. Implement searching of executables in $PATH
    c. Implement rc file loading at the initialization of shell
    d. Implement redirection (<, >) and pipe (|) s
    e. Implement viewing command line history with up and down arrow key
2. Build linux kernel from source code. Using Makefiles, create a graph of dependencies between original code and associated binaraies. The goal is to identify the effect of modification of a source file on a binary. Include intermediate results also. Graph visualization would be better. You can use go or python.
3. FUSE (File System in Userspace) allows programmers to create their own filesystem in userspace. Implement file transfer protocols such as FTP as a FUSE filesystem, and it should support basic functionality such as ls, cp, mount, umount, …
4. Develop a linux kernel module to monitor user activity in a multi-user environment. You can monitor some log files to get login information of the user. Also, write a userspace program (preffereble not in C) to communicate with kernel module and get the statistics and present it.
5. Evaluate performance of caching and paging (memory mountain)
6. Implement multi-threaded file server (a thread for each user) using any programming langauge (except python and nodejs). The server listens on TCP port 8888. And user communicates with it in JSON format consisting of 2 fields. An example request might look this this

    ```
    { “verb” : “GET”, “path”: “/example.txt” }
    ```

    There are 2 possible verbs, “GET” and “LIST”, both of which needs path. GET will retrieve the file at given path from the server, and LIST will return a list a files at given path.

    The reponse of the server is also JSON and should look like this

    ```
    {“error” : true, “what” : “Path not found”}
    ```

    In case of error. Otherwise,

    ```
    {“error”: false, “result”: “base64 encoded file”}
    ```

    For GET.

    ```
    {“error”: false, “result”: [ “file1”, “file2”, … ]}
    ```

    In case of LIST.

    At last, collabarate the team having problem number 3, to provide a FUSE interface for your server.
7. This problem needs Raspberry Pi (any version) along with some sensors. You have come up with an application (like temparature monitoring and logginge etc). You have to implement this application using GPIO programming using system calls in C only.
8. This problem consists of multiple communucating procecess using shared memory. These programs run one after another. Since there is no writing and reading the results file, there would be significant performance improvement. Following is the list of tasks that each program has to do.
    a. Read a csv file (of all floats, dimension will be given beforehand) to a shared memory region.

b. Apply log operation to each element of the shared memory
c. Find the min and max of each column and display that
d. Find the mean and standard deviation of each column and display that

Measure the computation time , io time in each program. Now, instead of shared memory, write intermediate results into disk (No need of Program a, program b has to write the results, which program c and d will read). Now measure computation time and IO time in each program. Visualize your observations.

9. Port a popular linux tool to Fuchisa OS (https://fuchsia.dev/) .