# README (LAB-2)

## PART-1

1)

a) A processor (CPU) is **the logic circuitry that responds to and processes the basic instructions that drive a computer**.
**A core, or CPU core, is the "brain" of a CPU**. It receives instructions, and performs calculations, or operations, to satisfy those instructions. A CPU can have multiple cores
The main difference between processor and core is that the processor is an electronic circuit inside the computer that carries out instructions to perform arithmetic, logical, control and input/output operations while the core is an execution unit inside the CPU that receives and executes instructions.

b) My machine has a 5 cpu core.
c) There are 12 processors in my machine.
d) The frequency of each processor is 1800 MHz.
e) The total physical memory the system has is 6386564 kB.
f) The total free memory in the system is 4287620 kB.
g) The total number of forks since last boot of the system is 16500
h) The system has performed 156723577 context-switches since bootup

2) The output is as follows :

Number of lines: 6
Number of words: 100
Numbers of bytes: 1079

3)

a) $ gcc cpu-print.c -o cpu-print $ ./cpu-print
By running the above two commands, we will get into an infinite loop after which we need to open another terminal and use the “ps -ef” command to find out the pid of the process spawned by the shell to run the cpu-print executable.

b) The PID of the parent of the cpu-print is detected and also its ancestors for 5 generations here is the filtered output.

c) $ ./cpu-print > /tmp/tmp.txt &
I/O redirection feature of the command line enables us to redirect the input and/or output of commands from and/or to files, or join multiple commands together using pipes to form what is known as a “command pipeline”.

Therefore, I/O redirection allows you to alter the input source of command as well as where its output and error messages are sent to. And this is made possible by the “<” and “>” redirection operators.

d) $ ./cpu-print | grep hello &

For the implementation of pipes in a bash shell :- A pipe is created using pipe(2), which returns two file descriptors, one referring to the read end of the pipe, the other referring to the write end. So we can see that from here the hello is written in the CMD.

## PART-2

To run the program, we need to do the following :

1. Download the zip folder provided.
2. Extract all the files along with the folder.
3. Open the folder and open the terminal in that folder.
4. Run the following commands:
    - $ gcc lab1.c -o lab1
    - ./lab1.c <file_name>

    (make sure that the file exists in the same directory)
    (test.txt file contains some of the commands for running in batch mode)
    After running the file, it asks for input from the user. There are two types of commands that can be run:
    The **internal commands** implemented for input are:
    ● ls ● pwd ● exit
    ● cd <directory>

    The **external commands** are general UNIX commands. External commands are of two types:
    1) Piped (example: $ sort <textfile_name> | uniq)
    2) Simple (example: $ sort <textfile_name>)
    Some of the errors are also handled in this code, which are:
    ➔ When wrong commands are entered, then the shell does not execute them.
    ➔ Wrong directory given along with cd command will be asked to correct
    ➔ Fork failures, execvp() failures and dup2() failures are handled

**Note: Use & after a command if you want it to run in the background.**

**Use Ctrl+C or exit to exit the shell, whereas Ctrl+D is used to produce an infinite loop.**