Hanish Patel

CS 497-01

Assignment 6

CS 497 – Assignment 6

**Abstract**

Examine system programming through Ubuntu with an application called C. Understand how processes work and how to view processes through the command line. Make system calls and understand the process between system calls and the kernel. Use basic file operations in C and create a Caesar cipher. Understand the command to direct the process in the command line. View information through the command line such as cpu %, memory usage, time, and more.

**Introduction**

We will use the built-in gedit to create a c file.

gedit [file]

We will compile the gcc with the following command

gcc -o executable cipher.c

We will run the c file by doing

./cfile [args] [filename]

And redirect with

>> “output file”

We will use -n to number lines,

-s to shift by n numbers, and

-r to reverse the shift direction

We will also use

ps [a][u][x]

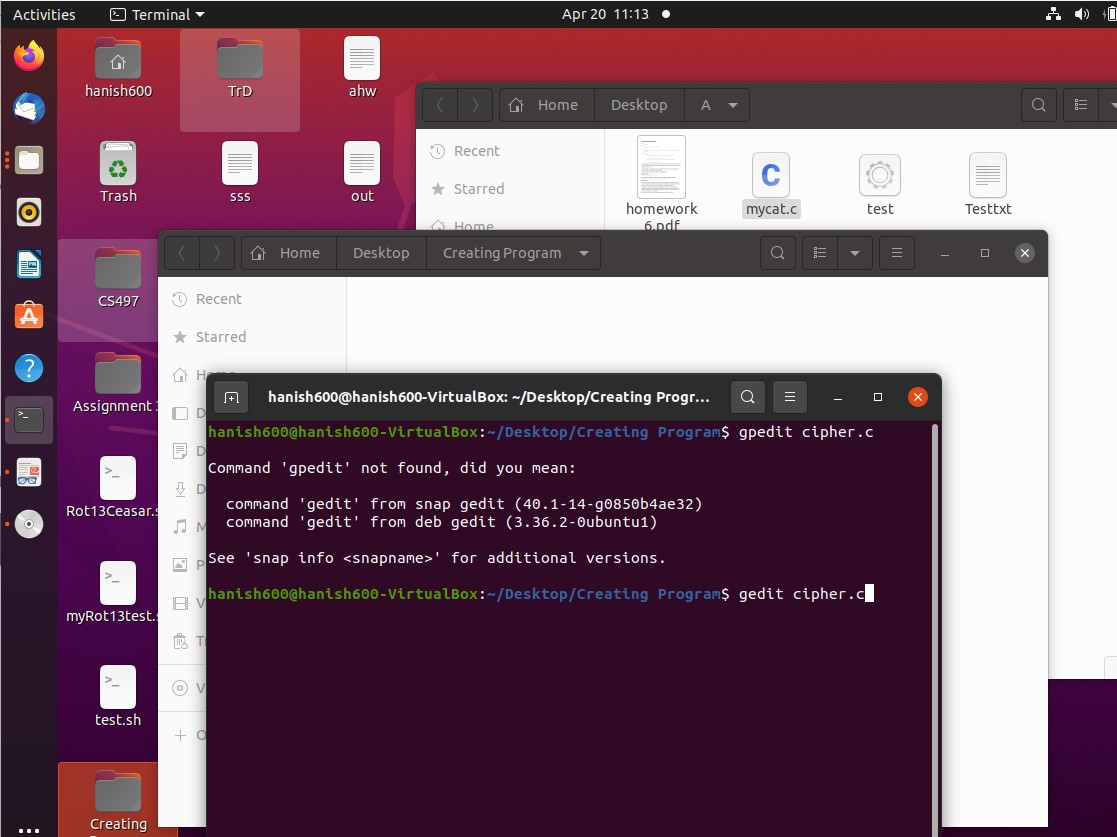
and

top -p [process id]

to view process information

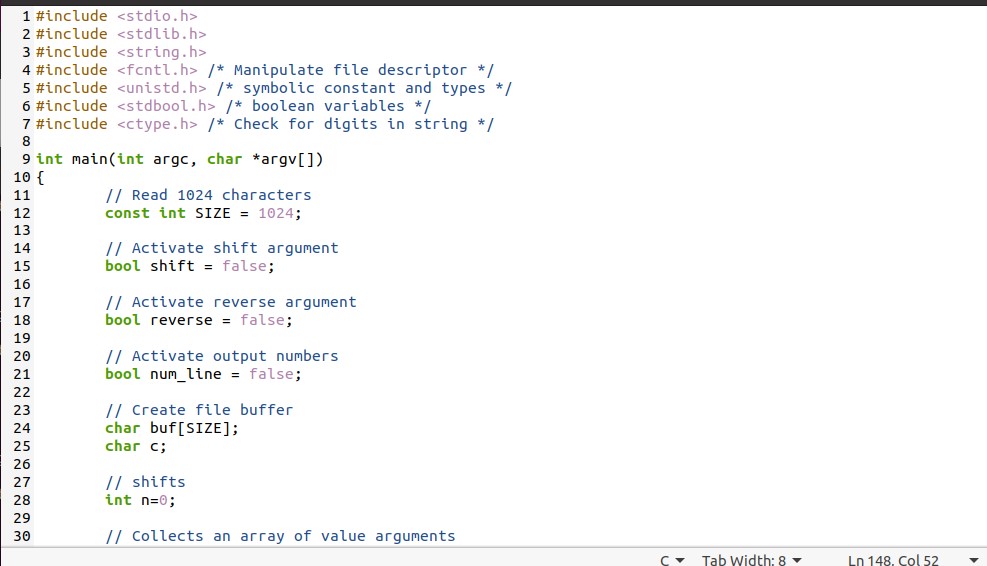
**Summary of Results**

Since we already have gcc installed, we open the editor using gedit cipher.c



We open up an editor file where we can edit our c file.

We start by listing out the important libraries we want to use.



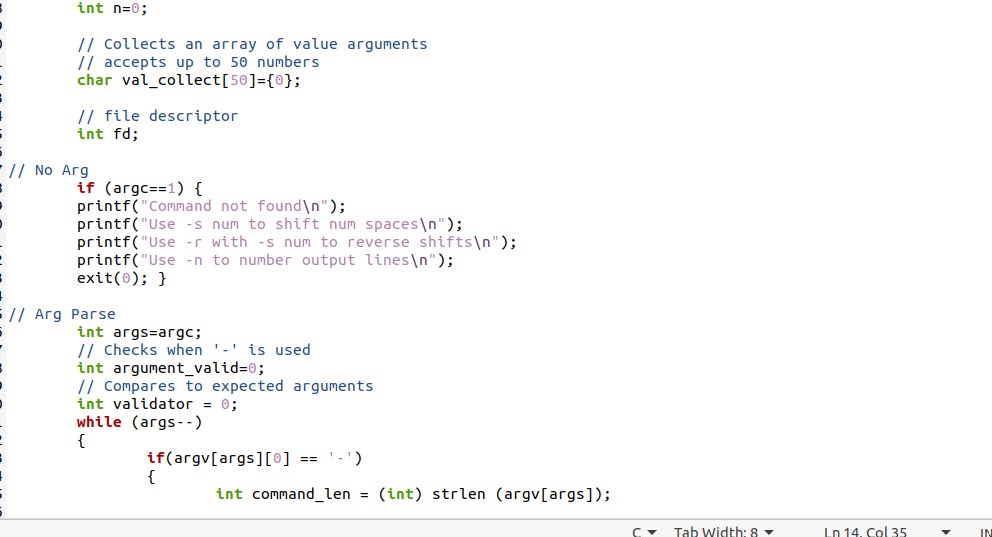
We make a main function that will take a arg counter and an array of commands.

We also make a const size and have 3 Boolean variables so we can adjust the arguments based on the users needs.

We also have a buffer that takes in 1024 file characters and inputs them into a single character c.

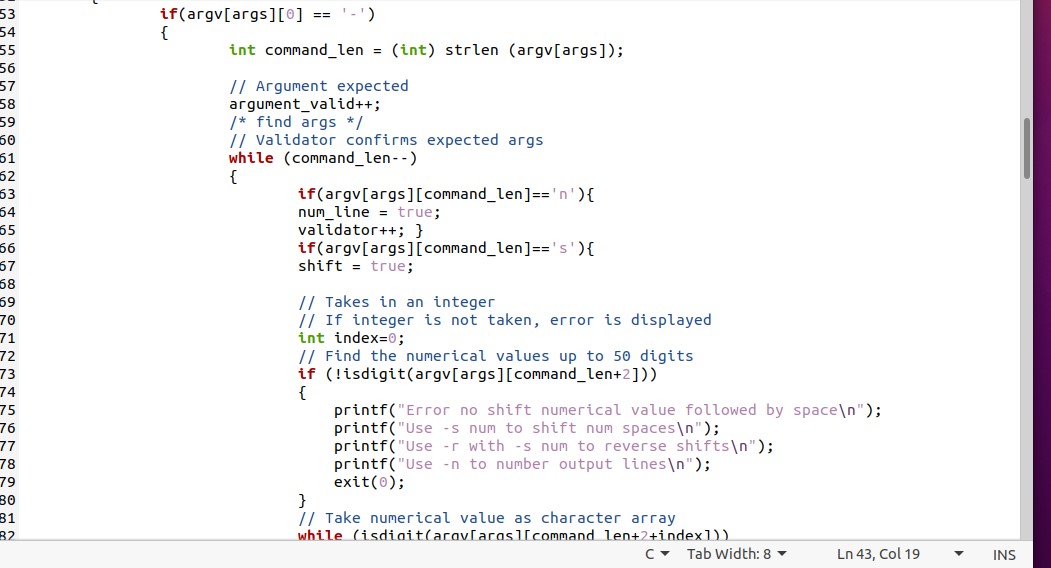
We take n shifts for the program to work.

Next, we make an array of characters to collect the integer values that were added to the command line. We take up to 50 integer characters as the shifts. We make our file descriptor and start with the conditions of no arguments and start with phasing different arguments.



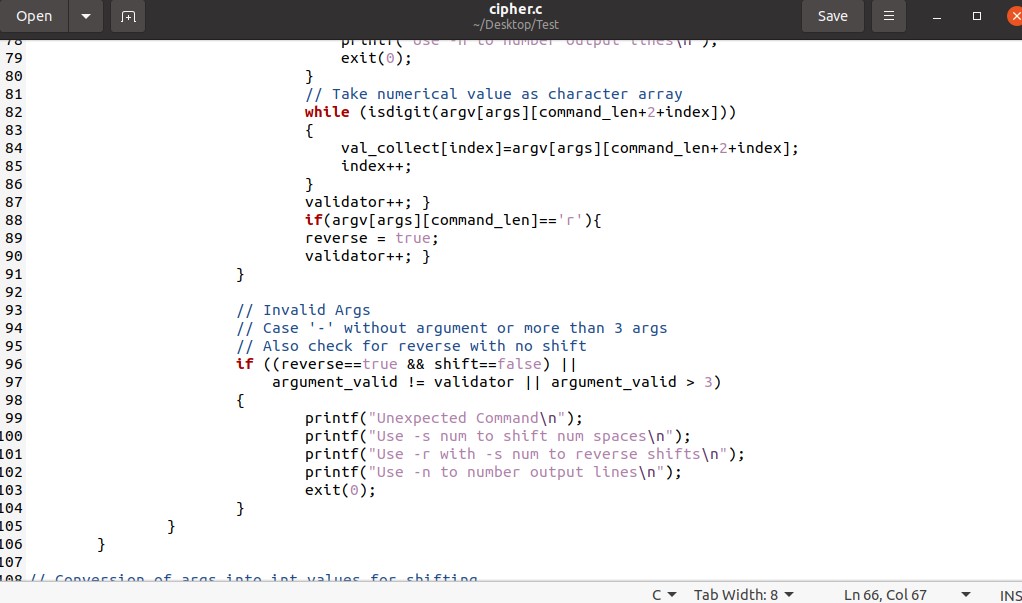
We take an argument valid and validator variable to check whether there will be more than 3 arguments or whether – is followed by an allowable character.

We immediately check if shift is followed by a space and integer values. We display the proper error or collect information needed for the program.



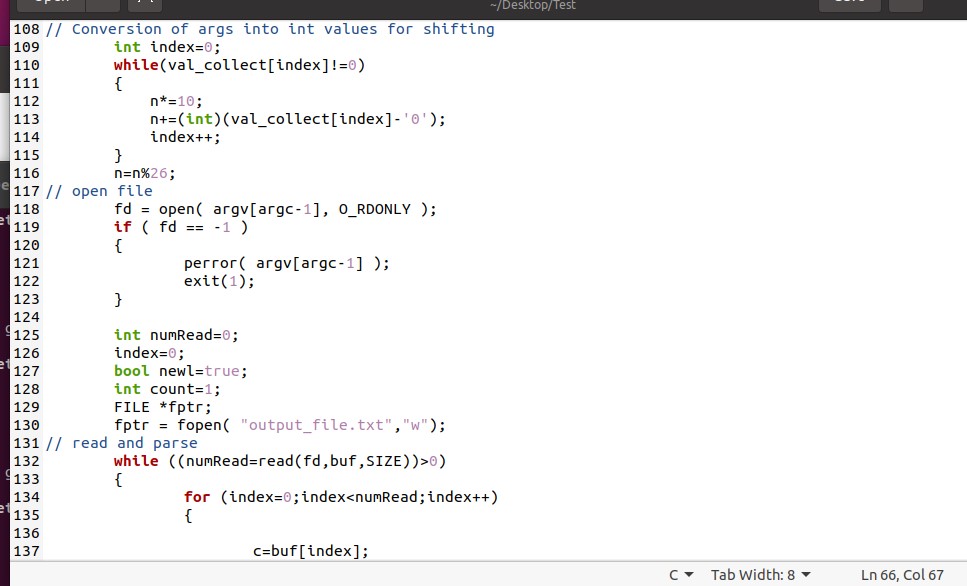
We make sure – is followed by allowable characters and there is no more than 3 total arguments.

We also check reverse is followed by a shift.

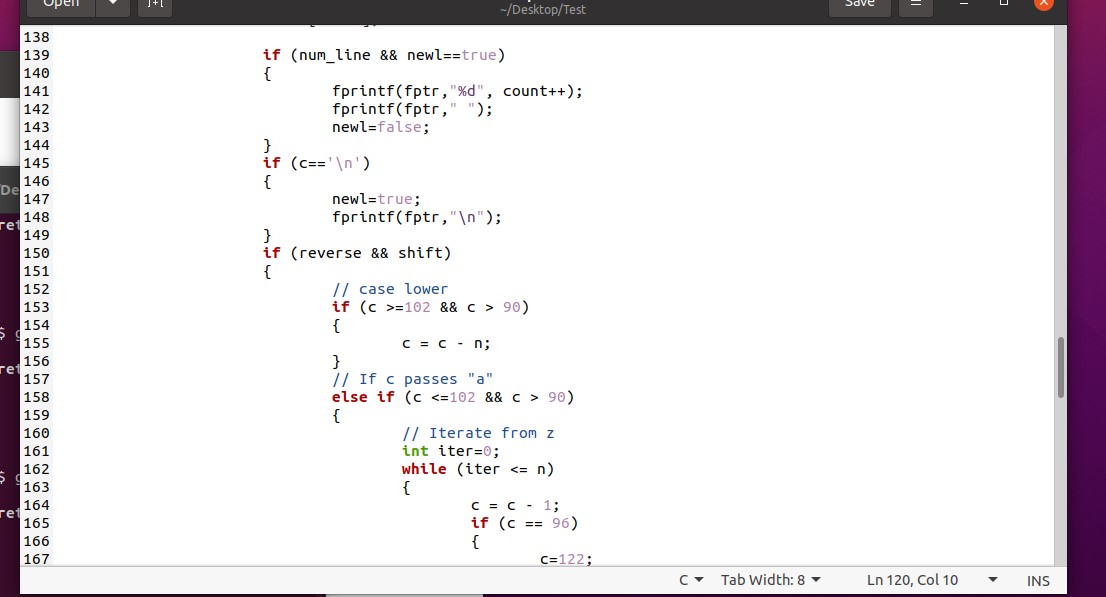


We convert the array of characters into an integer value.

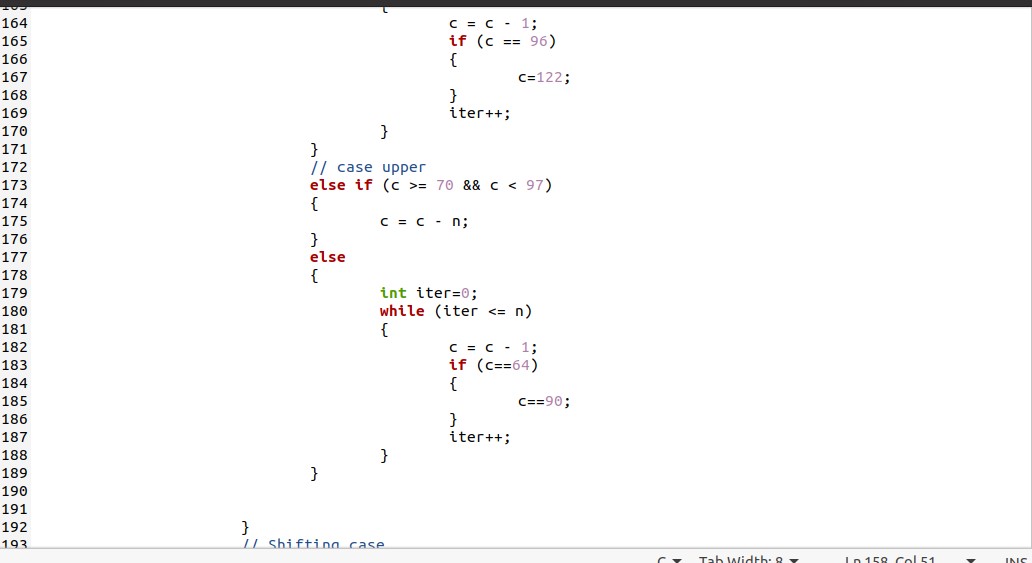
We also open the file with some variables and check any errors.

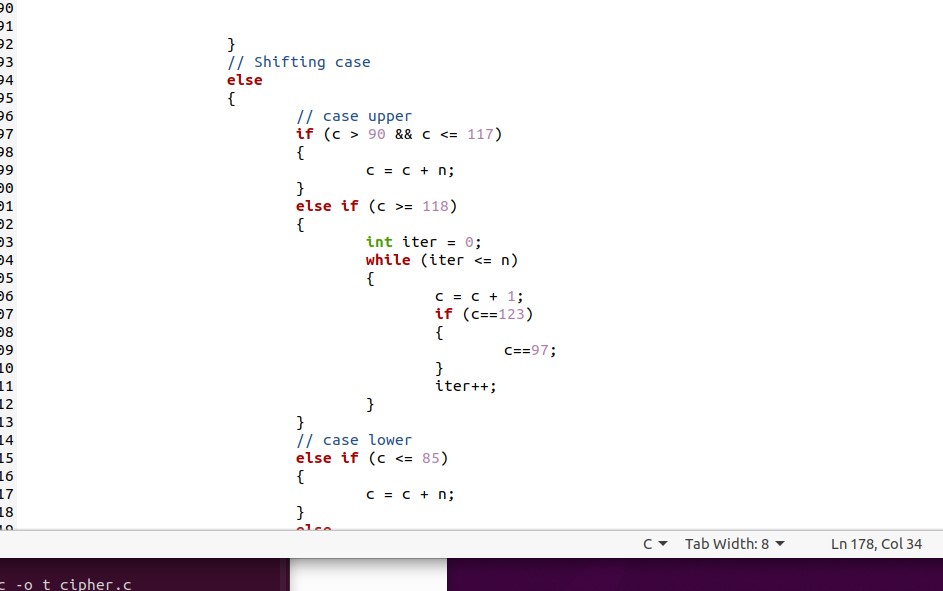


We fix in some conditions and later correct fprintf to be redirected from print.

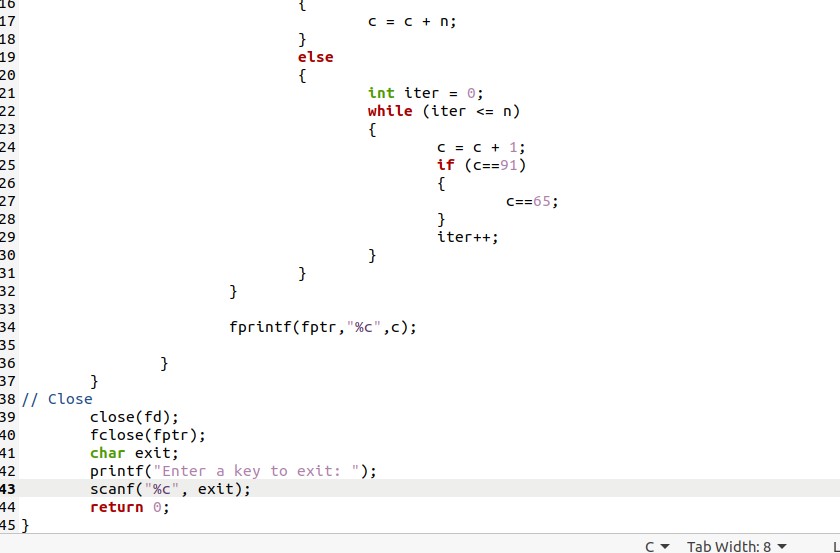


We also check the upper and lower case conditions where anything out of bound will automatically reset and be modded.

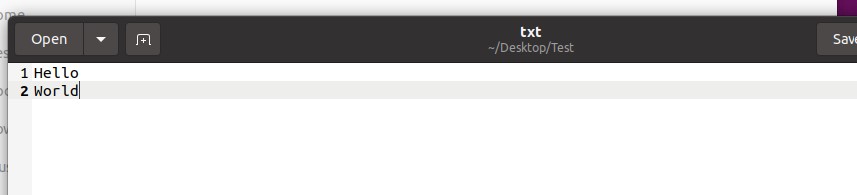




We later remove anything beyond the print statements from the scanning parts to ensure we get the proper standard output.

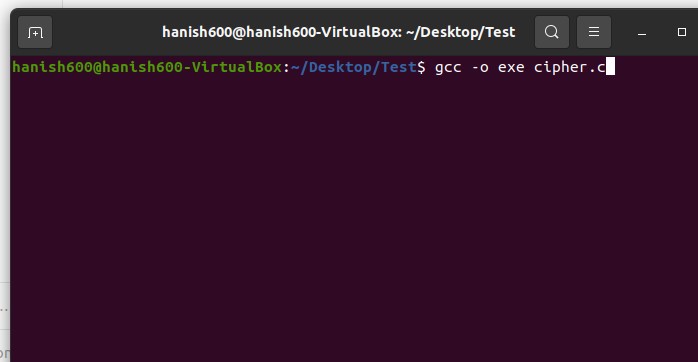


We have a basic text message.



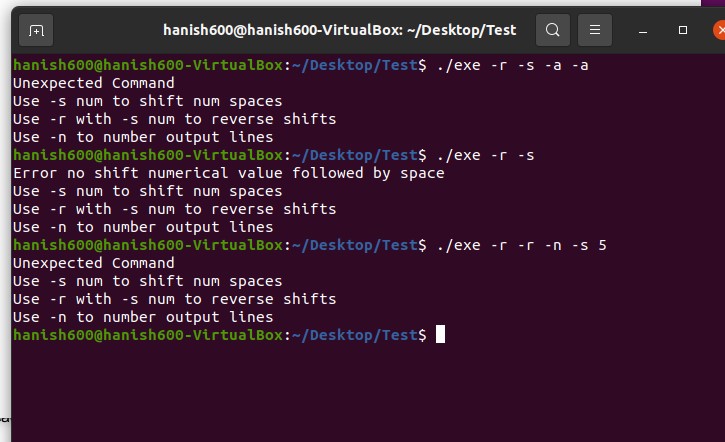
We fix our program by replacing fprintf with printf and removing the printf interaction with the user.

We compile the program to gcc and name it exe.



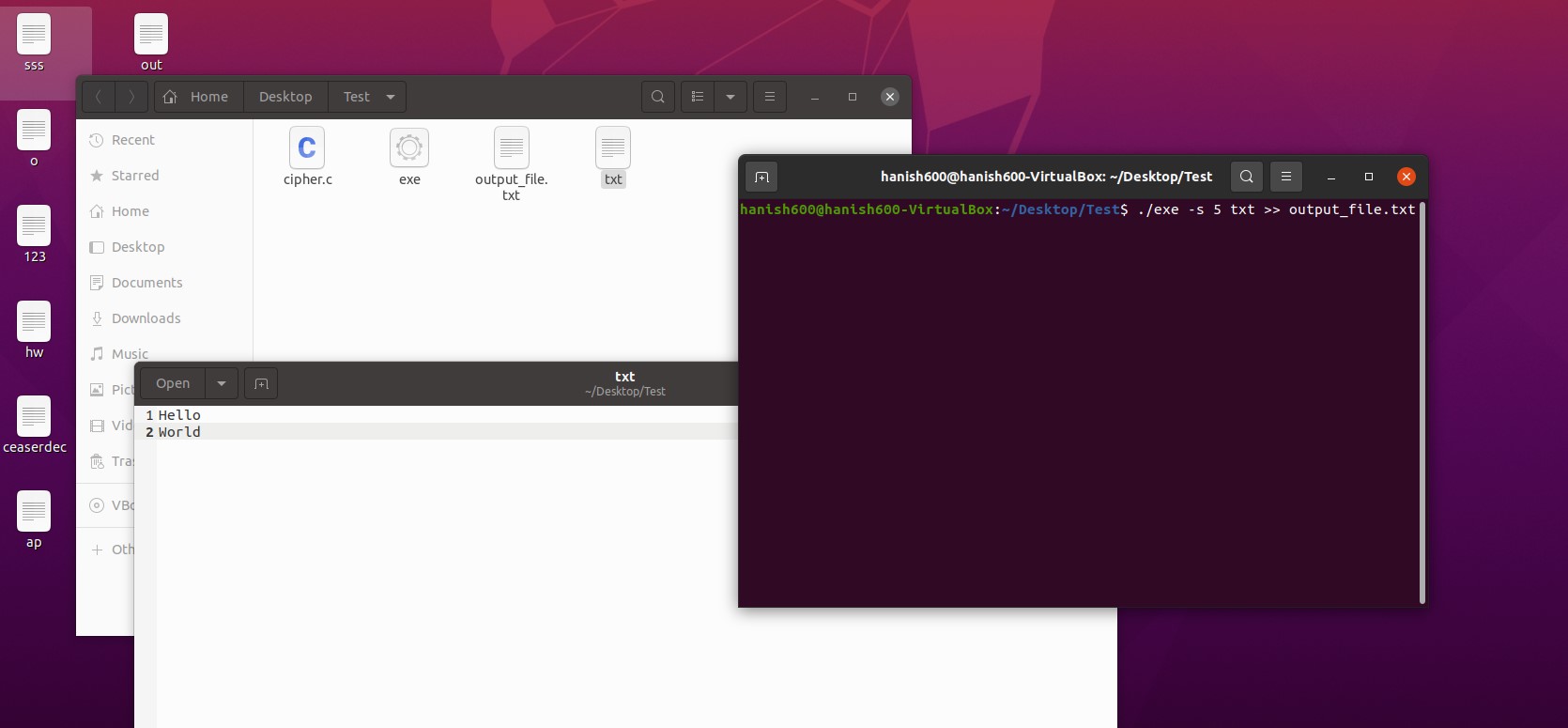
We start showing what are considered invalid prompts. We use ./exe with our arguments and filename.





As we see, these would all be invalid prompts. We use different error messages.

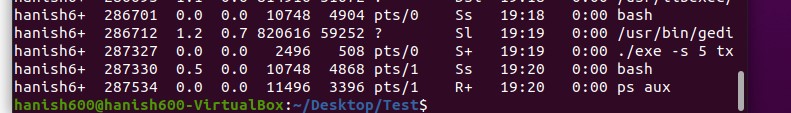
We direct a correct prompt to a standard output.



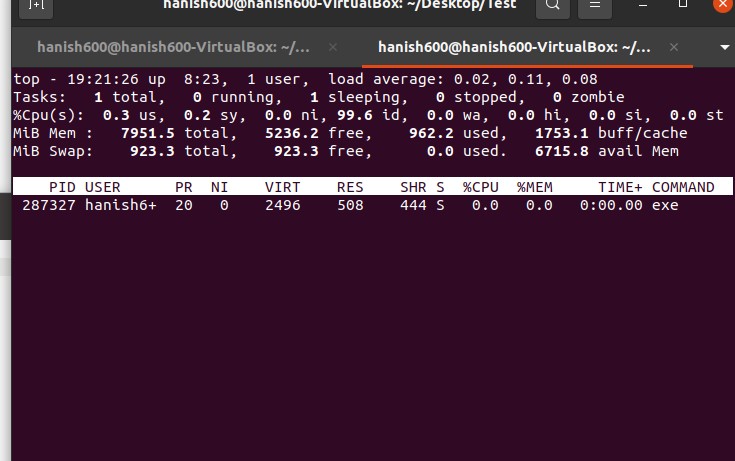
We get an output file.

We take the scanf break to scan our process.

We start with using ps u to see user processes.

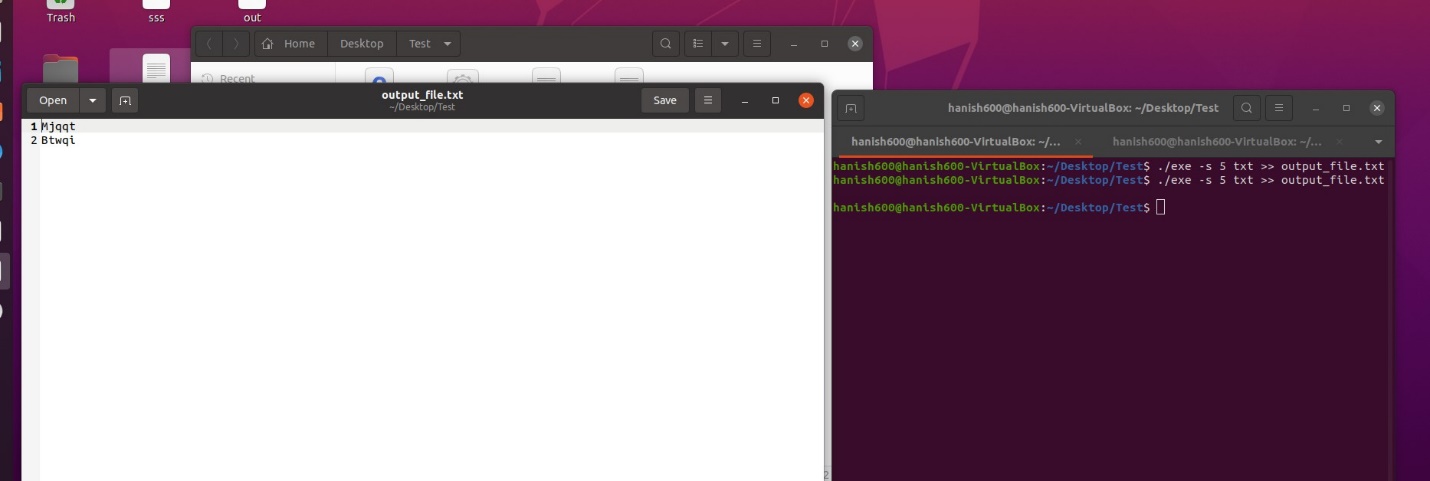


Then we examine this process more with top -p 287327.



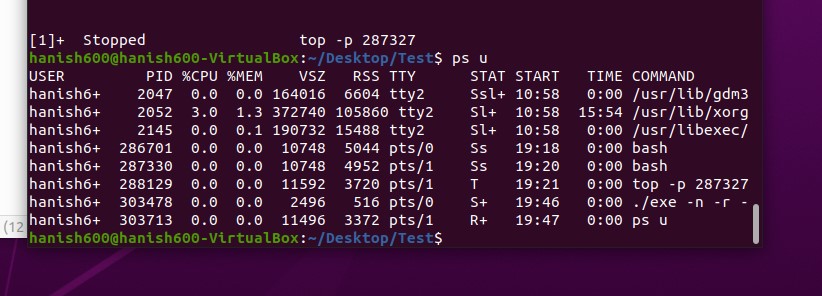
We can see how this process is being used.

We open our output file to get encrypted results.



We do this again this time with -n and -r to number and reverse our newly encrypted file.

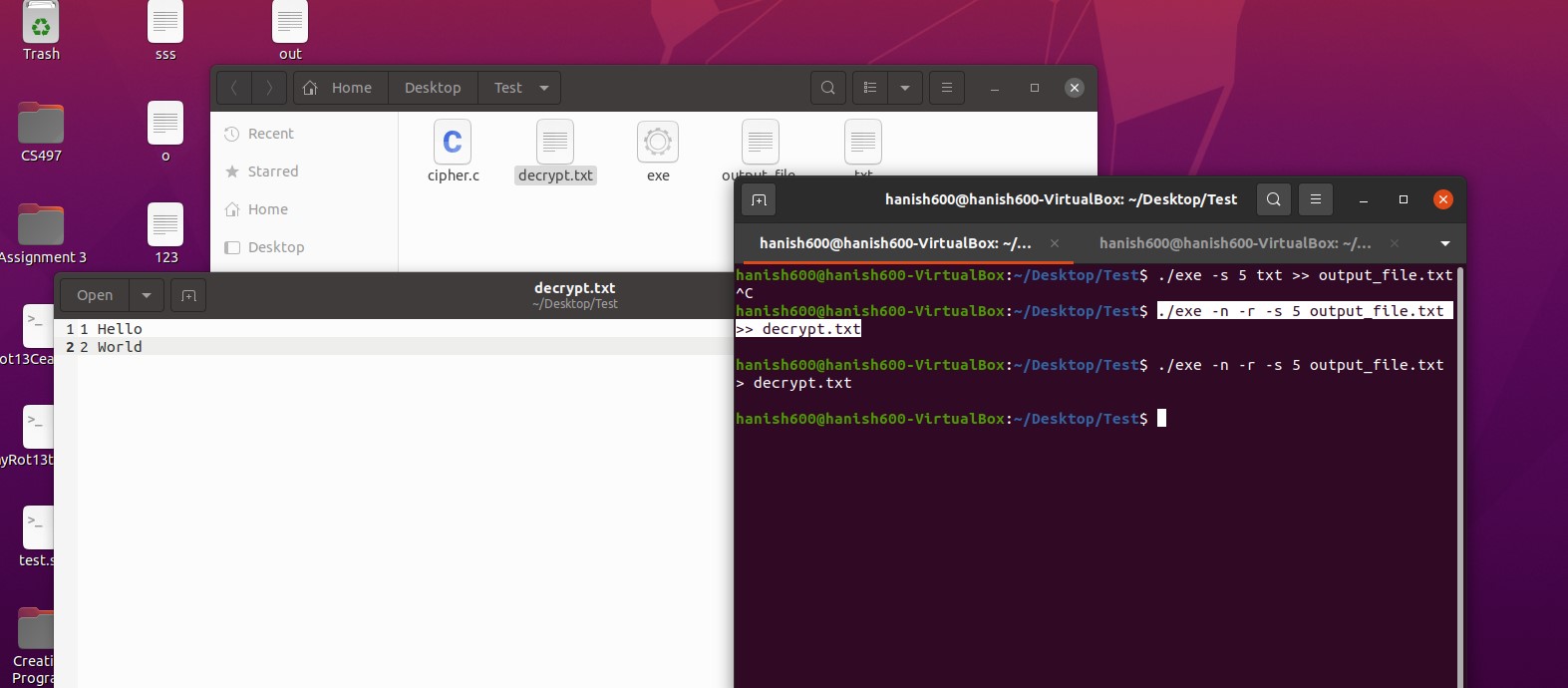
Before we do, we once again observe this process with ps u.



We see this is process 303478 and our top -p 287327 is keeping track of our old command run.

We can also use top -p on this command to observe it more. We just need the PID.

After some fixes, we reopen the decrypted file.



We see -n numbered our output and -r with -s shifts reversed our output by x shifts and > stored it in decrypt.txt

**Conclusion**

We found some information about our processes and how to view them using the command line. It was a great introduction to the syntax of the C language, and it was my first time going from C++ to C. Overall, it took a long time to do error checking and program the algorithm, but it was a fun activity. I learned a lot about C and the command output to a file. I also learned a lot about processes and how they work and appear in the view.

This information is useful to the user and an administrator so they can monitor processes that are not behaving. Our CPU % was 0, we were using 1753.1 memory in the buffer, it took 0.3 microseconds to execute, and it was a sleeping task. It was up for 19:21:26 at 8:23 PM. There was 1 user using the program. 0.2 microseconds were used by the system. This tells us a lot about the program. We would know if the program was using up too much of the system, or if it was being activate during a certain time, or if multiple users relied on the process. This would tell us a lot in how we would want to manage the process through the command line.