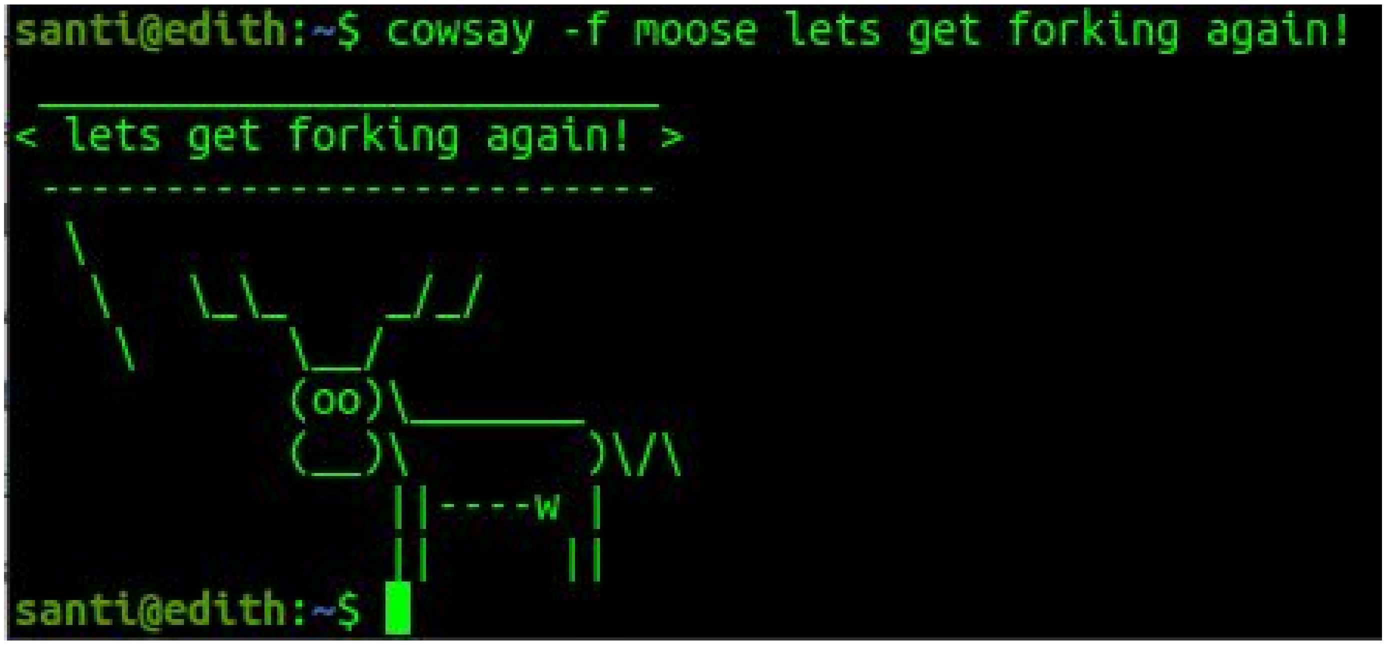


```
santi@edith:~$ cowsay -f moose lets get forking again!
 ____________________________
< lets get forking again! >
 ----------------------------
  \
   \   \_\_    _/_/
    \      \__/
           (oo)\_______
           (__)\       )\/\
               ||----w |
               ||     ||
santi@edith:~$
```

Fork Assignment - 3

COM301-P Assignment - 4

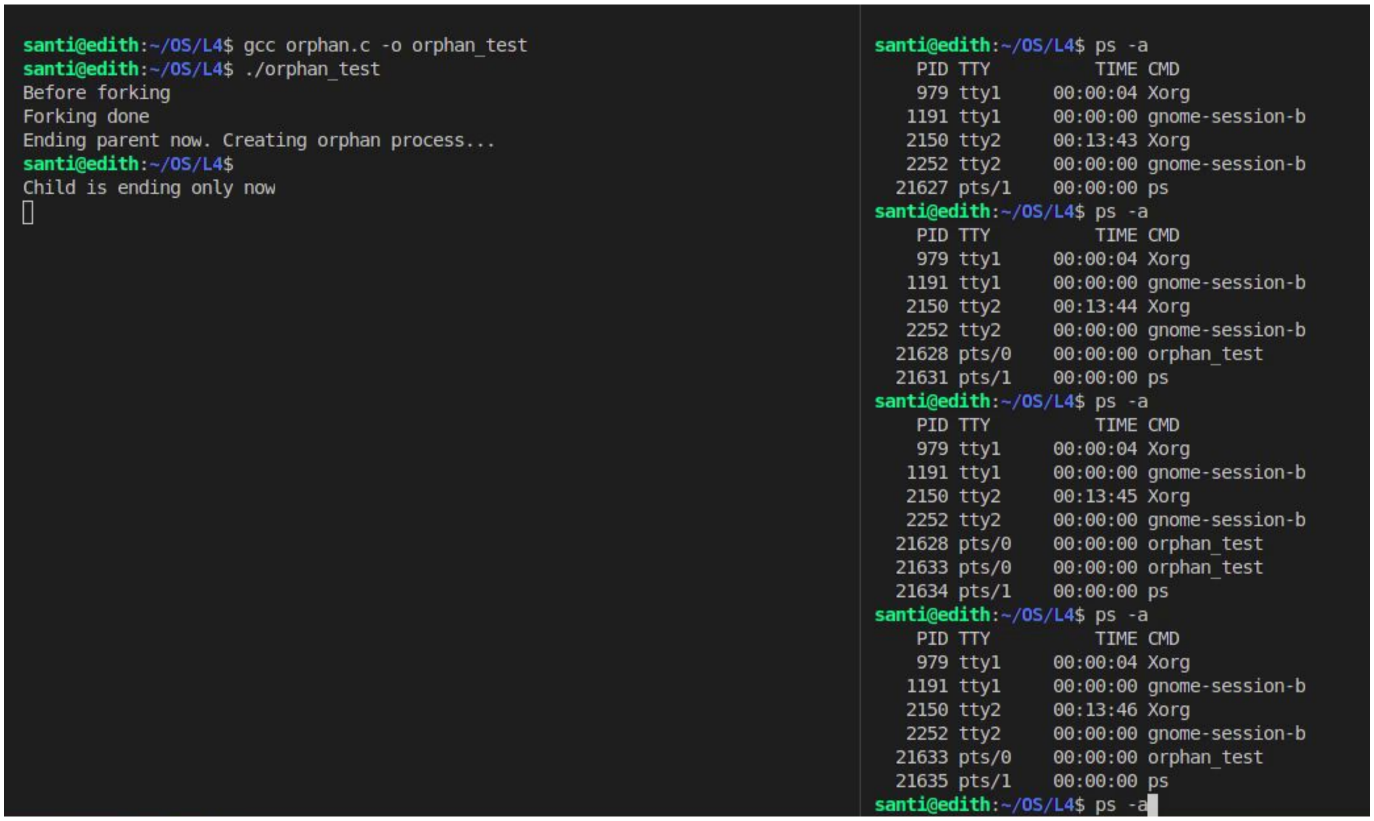
G S Santhosh Raghul
COE18B045

I have put only the outputs in this document. Please check the .c files for code. There are comments in the .c files for more clarity. I have also included the gcc command in the screenshots to indicate which file(s) correspond to which question.

Thank you!

1) Test drive a C program that creates Orphan and ZombieProcesses

Orphan Process

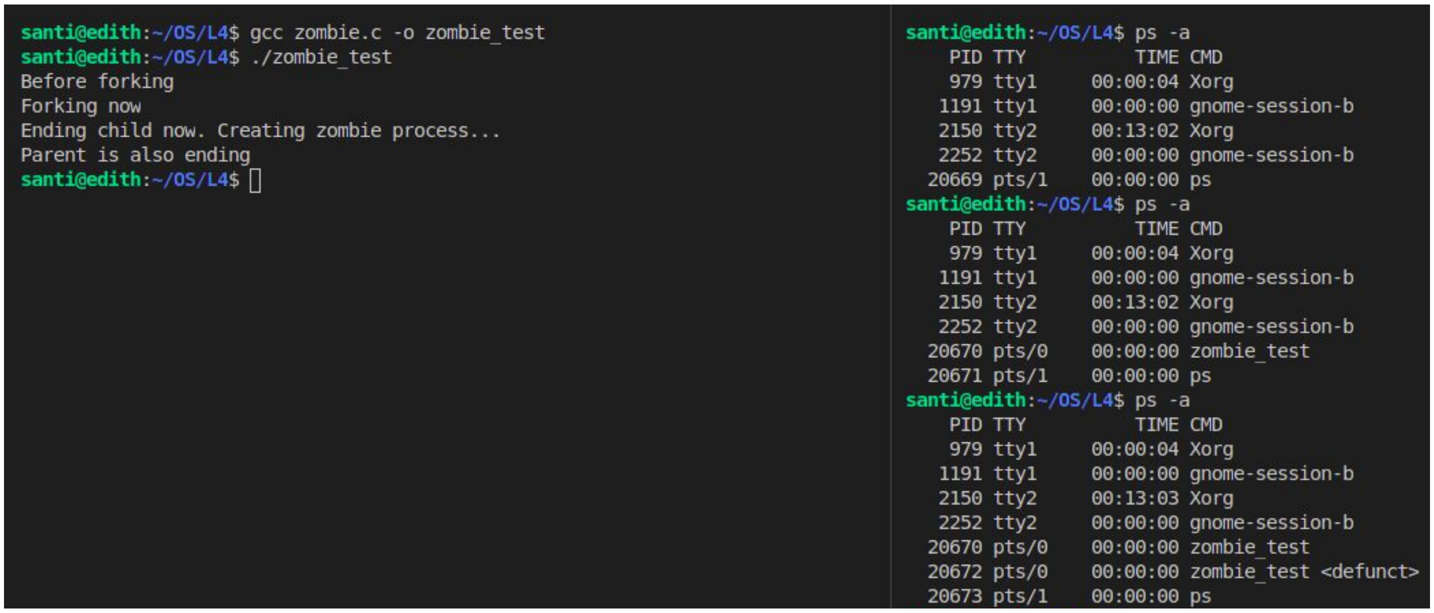


```
santi@edith:~/OS/L4$ gcc orphan.c -o orphan_test
santi@edith:~/OS/L4$ ./orphan_test
Before forking
Forking done
Ending parent now. Creating orphan process...
santi@edith:~/OS/L4$
Child is ending only now
```

```
santi@edith:~/OS/L4$ ps -a
    PID TTY          TIME CMD
    979 tty1     00:00:04 Xorg
   1191 tty1     00:00:00 gnome-session-b
   2150 tty2     00:13:43 Xorg
   2252 tty2     00:00:00 gnome-session-b
  21627 pts/1    00:00:00 ps
santi@edith:~/OS/L4$ ps -a
    PID TTY          TIME CMD
    979 tty1     00:00:04 Xorg
   1191 tty1     00:00:00 gnome-session-b
   2150 tty2     00:13:44 Xorg
   2252 tty2     00:00:00 gnome-session-b
  21628 pts/0    00:00:00 orphan_test
  21631 pts/1    00:00:00 ps
santi@edith:~/OS/L4$ ps -a
    PID TTY          TIME CMD
    979 tty1     00:00:04 Xorg
   1191 tty1     00:00:00 gnome-session-b
   2150 tty2     00:13:45 Xorg
   2252 tty2     00:00:00 gnome-session-b
  21628 pts/0    00:00:00 orphan_test
  21633 pts/0    00:00:00 orphan_test
  21634 pts/1    00:00:00 ps
santi@edith:~/OS/L4$ ps -a
    PID TTY          TIME CMD
    979 tty1     00:00:04 Xorg
   1191 tty1     00:00:00 gnome-session-b
   2150 tty2     00:13:46 Xorg
   2252 tty2     00:00:00 gnome-session-b
  21633 pts/0    00:00:00 orphan_test
  21635 pts/1    00:00:00 ps
santi@edith:~/OS/L4$ ps -a
```

- The first `ps -a` is before running the program
- The second `ps -a` is before forking. The pid is 21628.
- The third `ps -a` is after forking. Parent pid is 21628 and child pid is 21633.
- In the fourth `ps -a`, the parent has terminated and only the child with pid 21633 is active ⇒ **orphan process**.

Zombie Process



```
santi@edith:~/OS/L4$ gcc zombie.c -o zombie_test
santi@edith:~/OS/L4$ ./zombie_test
Before forking
Forking now
Ending child now. Creating zombie process...
Parent is also ending
santi@edith:~/OS/L4$
```

```
santi@edith:~/OS/L4$ ps -a
    PID TTY          TIME CMD
    979 tty1     00:00:04 Xorg
   1191 tty1     00:00:00 gnome-session-b
   2150 tty2     00:13:02 Xorg
   2252 tty2     00:00:00 gnome-session-b
  20669 pts/1    00:00:00 ps
santi@edith:~/OS/L4$ ps -a
    PID TTY          TIME CMD
    979 tty1     00:00:04 Xorg
   1191 tty1     00:00:00 gnome-session-b
   2150 tty2     00:13:02 Xorg
   2252 tty2     00:00:00 gnome-session-b
  20670 pts/0    00:00:00 zombie_test
  20671 pts/1    00:00:00 ps
santi@edith:~/OS/L4$ ps -a
    PID TTY          TIME CMD
    979 tty1     00:00:04 Xorg
   1191 tty1     00:00:00 gnome-session-b
   2150 tty2     00:13:03 Xorg
   2252 tty2     00:00:00 gnome-session-b
  20670 pts/0    00:00:00 zombie_test
  20672 pts/0    00:00:00 zombie_test <defunct>
  20673 pts/1    00:00:00 ps
```

- The first `ps -a` is before running the program
- The second `ps -a` is before forking. The pid is 20670.
- The third `ps -a` is after forking and the termination of the child. Parent pid is 20670 and child pid is 20672. Here, the child has terminated and yet it is shown to be still active as a `<defunct>` (zombie) process since the parent has not yet terminated ⇒ **zombie process**.

2) Develop a multiprocessing version of Merge or Quick Sort. Extra credits would be given for those who implement both in a multiprocessing fashion [increased no of processes to enhance the effect of parallelization]

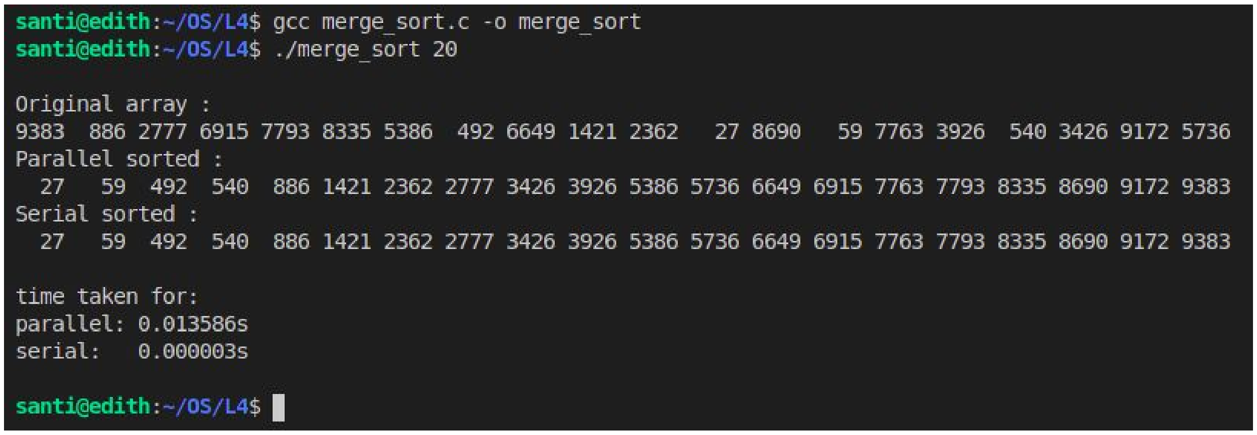
Explanation

Same for both mergesort and quicksort. The programs do the following:

- Get the number of array elements from the command line argument.
- Create a shm array (for parallel) and a normal array (for serial) and fill the array with random numbers. (both array contain the same numbers)
- Print original array
- Sort parallel and calculate runtime and display the sorted array
- Sort serial and calculate runtime and display the sorted array
- Print time taken for both parallel and serial sorting

Note: `rand()` can be replaced with `scanf()` for user input instead of random array

Output - both mergesort and quicksort

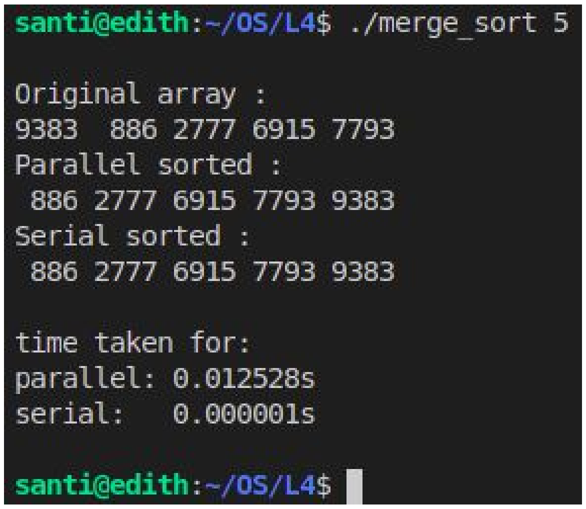


```
santi@edith:~/OS/L4$ gcc merge_sort.c -o merge_sort
santi@edith:~/OS/L4$ ./merge_sort 20

Original array :
9383  886 2777 6915 7793 8335 5386  492 6649 1421 2362   27 8690   59 7763 3926  540 3426 9172 5736
Parallel sorted :
  27   59  492  540  886 1421 2362 2777 3426 3926 5386 5736 6649 6915 7763 7793 8335 8690 9172 9383
Serial sorted :
  27   59  492  540  886 1421 2362 2777 3426 3926 5386 5736 6649 6915 7763 7793 8335 8690 9172 9383

time taken for:
parallel: 0.013586s
serial:   0.000003s

santi@edith:~/OS/L4$ 
```

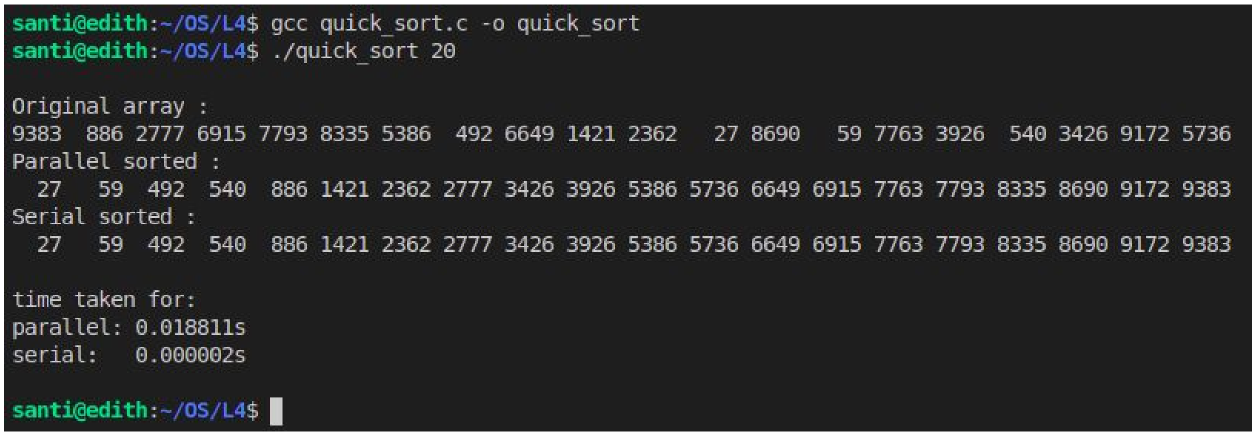


```
santi@edith:~/OS/L4$ ./merge_sort 5

Original array :
9383  886 2777 6915 7793
Parallel sorted :
 886 2777 6915 7793 9383
Serial sorted :
 886 2777 6915 7793 9383

time taken for:
parallel: 0.012528s
serial:   0.000001s

santi@edith:~/OS/L4$
```

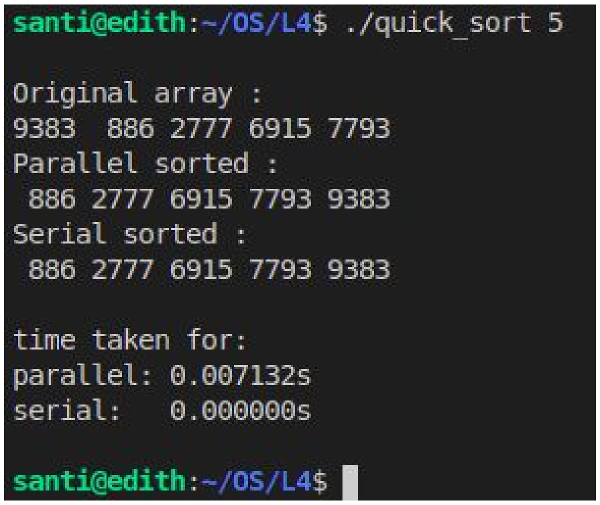


```
santi@edith:~/OS/L4$ gcc quick_sort.c -o quick_sort
santi@edith:~/OS/L4$ ./quick_sort 20

Original array :
9383  886 2777 6915 7793 8335 5386  492 6649 1421 2362   27 8690   59 7763 3926  540 3426 9172 5736
Parallel sorted :
  27   59  492  540  886 1421 2362 2777 3426 3926 5386 5736 6649 6915 7763 7793 8335 8690 9172 9383
Serial sorted :
  27   59  492  540  886 1421 2362 2777 3426 3926 5386 5736 6649 6915 7763 7793 8335 8690 9172 9383

time taken for:
parallel: 0.018811s
serial:   0.000002s

santi@edith:~/OS/L4$
```



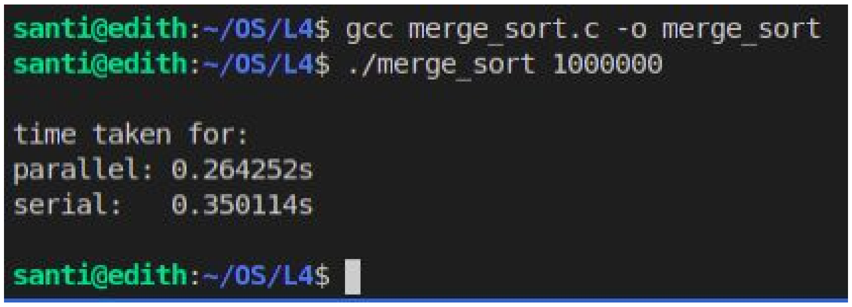
```
santi@edith:~/OS/L4$ ./quick_sort 5

Original array :
9383  886 2777 6915 7793
Parallel sorted :
 886 2777 6915 7793 9383
Serial sorted :
 886 2777 6915 7793 9383

time taken for:
parallel: 0.007132s
serial:   0.000000s

santi@edith:~/OS/L4$
```

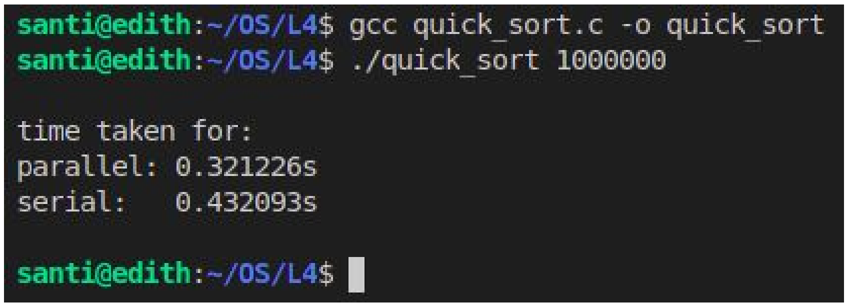
As we can see, time for parallel is more. This is because, time taken to create new processes is more than the execution time itself. Only when I used **serial sort for array size < 10000 and parallel sort for larger arrays** (I also commented the lines that print the array) and used an **array size of 1 million**, I got lesser run time for parallel than serial.



```
santi@edith:~/OS/L4$ gcc merge_sort.c -o merge_sort
santi@edith:~/OS/L4$ ./merge_sort 1000000

time taken for:
parallel: 0.264252s
serial:   0.350114s

santi@edith:~/OS/L4$ 
```



```
santi@edith:~/OS/L4$ gcc quick_sort.c -o quick_sort
santi@edith:~/OS/L4$ ./quick_sort 1000000

time taken for:
parallel: 0.321226s
serial:   0.432093s

santi@edith:~/OS/L4$ 
```

3) Develop a C program to count the maximum number of processes that can be created using fork call.

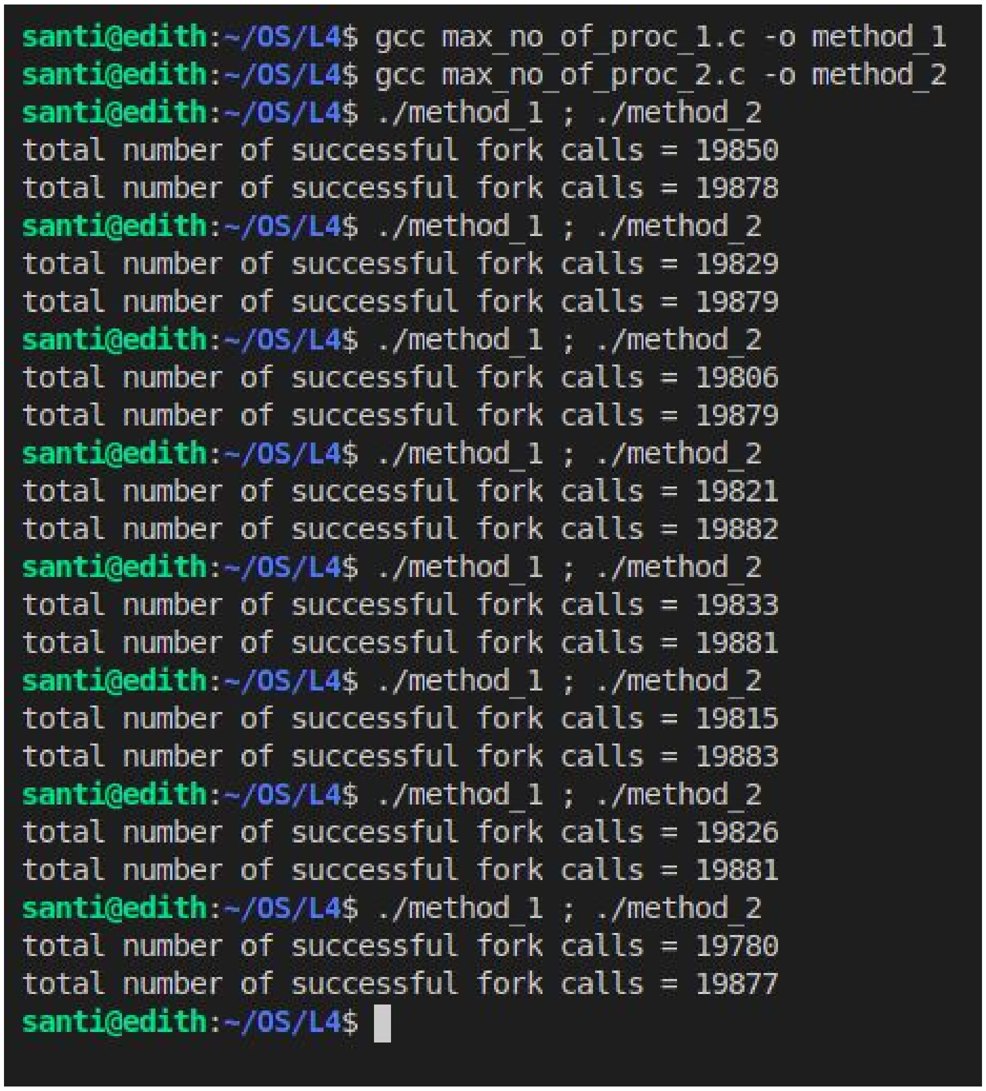
Method 1

Create a shared memory variable. Fork inside a loop and increment the shared variable by 1 in all the successful fork calls. Break out of the loop when a fork call returns -1. Finally print the count value.

Method 2

Find the maximum number of user processes allowed using `ulimit -u` Then run a for loop for i = 0, i < max and call `fork()` in the loop. For every `fork()` call, if it is successful, put `exit(1)` in the child block. Now use another for loop (after the previous for loop) and get the exit status of all the child processes using `exit(&status)` and keep adding the exit value to a counter variable. For unsuccessful forks, the value of exit status will be 0, for successful calls it will be 1. So, the final value of the counter variable will be the no of successful fork calls.

Output



```
santi@edith:~/OS/L4$ gcc max_no_of_proc_1.c -o method_1
santi@edith:~/OS/L4$ gcc max_no_of_proc_2.c -o method_2
santi@edith:~/OS/L4$ ./method_1 ; ./method_2
total number of successful fork calls = 19850
total number of successful fork calls = 19878
santi@edith:~/OS/L4$ ./method_1 ; ./method_2
total number of successful fork calls = 19829
total number of successful fork calls = 19879
santi@edith:~/OS/L4$ ./method_1 ; ./method_2
total number of successful fork calls = 19806
total number of successful fork calls = 19879
santi@edith:~/OS/L4$ ./method_1 ; ./method_2
total number of successful fork calls = 19821
total number of successful fork calls = 19882
santi@edith:~/OS/L4$ ./method_1 ; ./method_2
total number of successful fork calls = 19833
total number of successful fork calls = 19881
santi@edith:~/OS/L4$ ./method_1 ; ./method_2
total number of successful fork calls = 19815
total number of successful fork calls = 19883
santi@edith:~/OS/L4$ ./method_1 ; ./method_2
total number of successful fork calls = 19826
total number of successful fork calls = 19881
santi@edith:~/OS/L4$ ./method_1 ; ./method_2
total number of successful fork calls = 19780
total number of successful fork calls = 19877
santi@edith:~/OS/L4$ 
```

4) Develop your own command shell [say mark it with @] that accepts user commands (System or User Binaries), executes the commands and returns the prompt for further user interaction. Also extend this to support a history feature (if the user types !6 at the command prompt; it should display the most recent execute 6 commands). You may provide validation features such as !10 when there are only 9 files to display the entire history contents and other validations required for the history feature;

Explanation

The program first gets the home directory of the user and stores it in a variable for later use. Then in an infinite loop, the following things happen:

- Get current working directory, replace home directory with "~" if found and display the prompt. Now it waits for the user to enter a command.
- Get the command from the user, store it in a linked list (for history purpose) and call `run_command()` function

The `run_command()` function does the following:

- Processes the command and splits it into command, arguments and puts it into `args[]` variable.
- The commands - `cd` `exit` and `!n` for history are handled separately with an if block.
    - If the command starts with `!`, `history()` function is called which is followed by a return statement.
    - If the command is `cd`, the actual `cd` linux command is not executed. Instead, `change_directory()` function is called which is followed by a return statement.
    - If the command is exit, `exit(0)` is called and the program ends.

- Now, if the function has not returned yet, we fork to execute the command. `wait(NULL);` is called in the parent block and the command the user entered is called using execvp in the child block. If the exec fails, an error message is displayed and the child is exited.

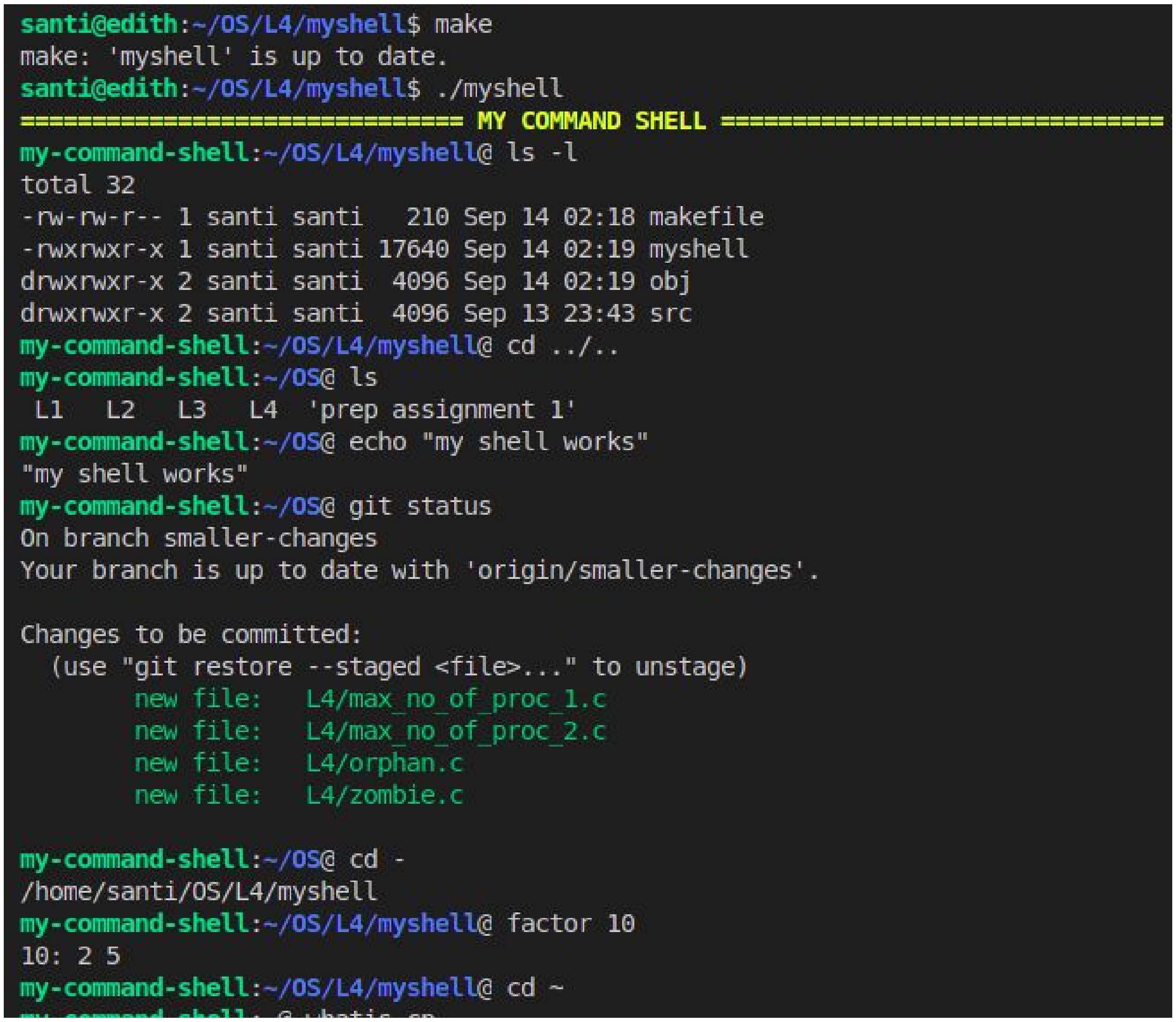
The `history()` function does the following:

- If ! is not followed by a number, report error.
- If the given number is more than the number of commands executed, report error.
- Else, print the commands that have been executed so far. Here, I have used the `history` command's style. The command number is also displayed.

The `change_directory()` function does the following:

- As I said, the linux command `cd` is not used, instead, the c function `chdir()` has been used.
- If there is no 2nd argument or if the 2nd argument is "`~`", go to the home directory.
- If the 2nd argument is "`-`" :
  - Check if the previous directory is set. If so, go to the previous directory.
  - Else, report error.
- Change the directory to whatever the user has specified.
  - If changing the directory was unsuccessful, report error.
  - Else, set previous directory to what it was changed form

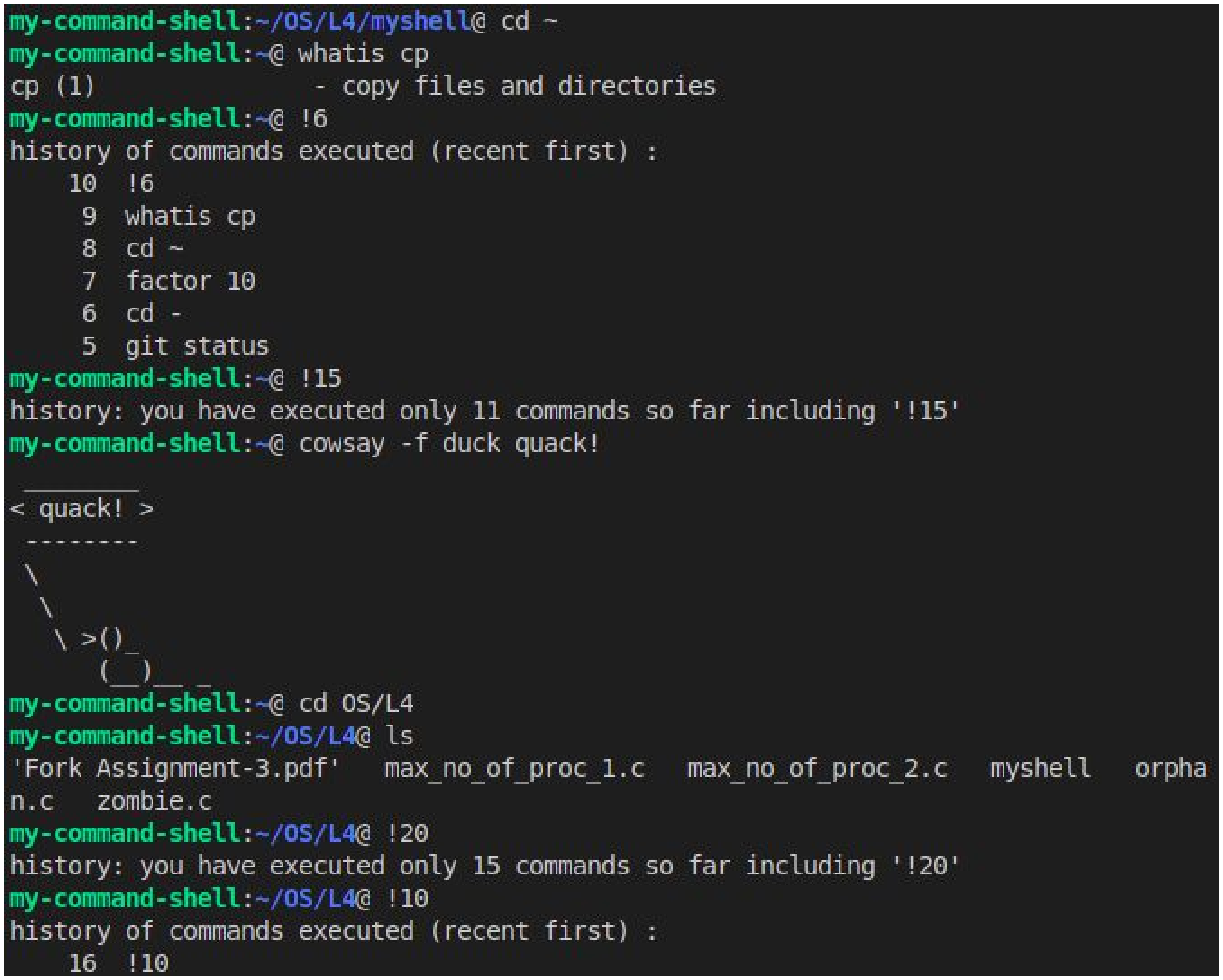
Output



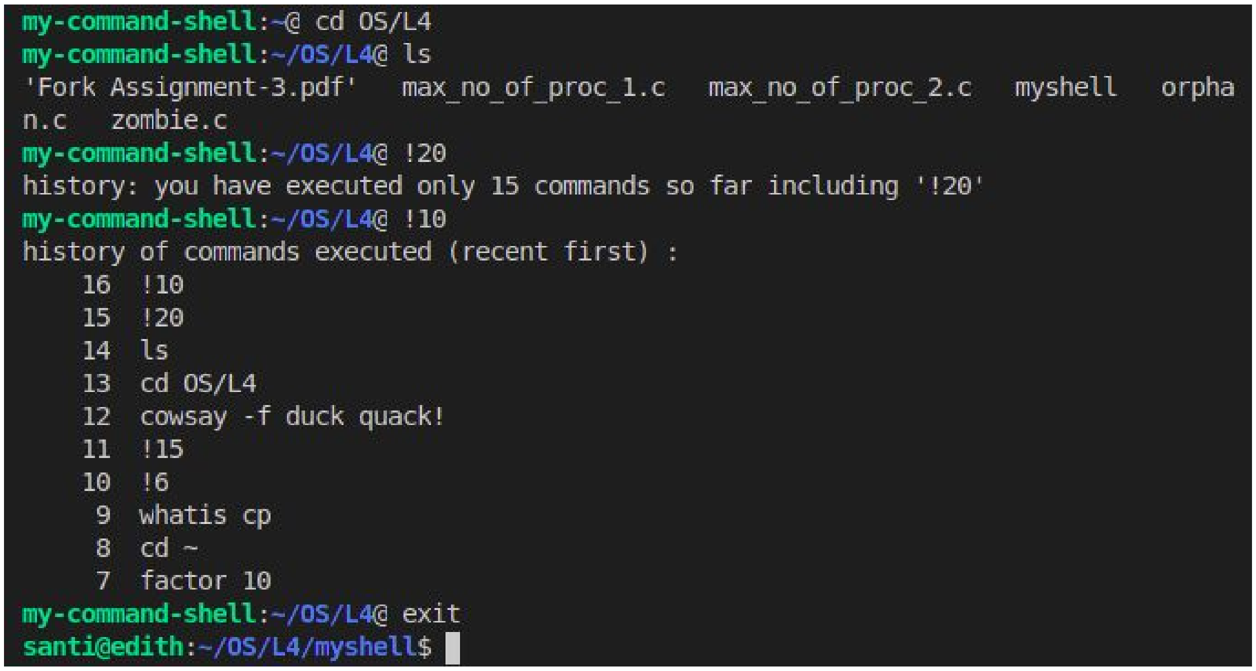
```
santi@edith:~/OS/L4/myshell$ make
make: 'myshell' is up to date.
santi@edith:~/OS/L4/myshell$ ./myshell
=============================== MY COMMAND SHELL ===============================
my-command-shell:~/OS/L4/myshell@ ls -l
total 32
-rw-rw-r-- 1 santi santi   210 Sep 14 02:18 makefile
-rwxrwxr-x 1 santi santi 17640 Sep 14 02:19 myshell
drwxrwxr-x 2 santi santi  4096 Sep 14 02:19 obj
drwxrwxr-x 2 santi santi  4096 Sep 13 23:43 src
my-command-shell:~/OS/L4/myshell@ cd ../..
my-command-shell:~/OS@ ls
 L1   L2   L3   L4  'prep assignment 1'
my-command-shell:~/OS@ echo "my shell works"
"my shell works"
my-command-shell:~/OS@ git status
On branch smaller-changes
Your branch is up to date with 'origin/smaller-changes'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   L4/max_no_of_proc_1.c
        new file:   L4/max_no_of_proc_2.c
        new file:   L4/orphan.c
        new file:   L4/zombie.c

my-command-shell:~/OS@ cd -
/home/santi/OS/L4/myshell
my-command-shell:~/OS/L4/myshell@ factor 10
10: 2 5
my-command-shell:~/OS/L4/myshell@ cd ~
my-command-shell:~@ whatis cp
```



```
my-command-shell:~/OS/L4/myshell@ cd ~
my-command-shell:~@ whatis cp
cp (1)               - copy files and directories
my-command-shell:~@ !6
history of commands executed (recent first) :
    10  !6
     9  whatis cp
     8  cd ~
     7  factor 10
     6  cd -
     5  git status
my-command-shell:~@ !15
history: you have executed only 11 commands so far including '!15'
my-command-shell:~@ cowsay -f duck quack!
 ________
< quack! >
 --------
 \
  \
   \ >()_
      (__)__ _
my-command-shell:~@ cd OS/L4
my-command-shell:~/OS/L4@ ls
'Fork Assignment-3.pdf'   max_no_of_proc_1.c   max_no_of_proc_2.c   myshell   orpha
n.c   zombie.c
my-command-shell:~/OS/L4@ !20
history: you have executed only 15 commands so far including '!20'
my-command-shell:~/OS/L4@ !10
history of commands executed (recent first) :
    16  !10
```

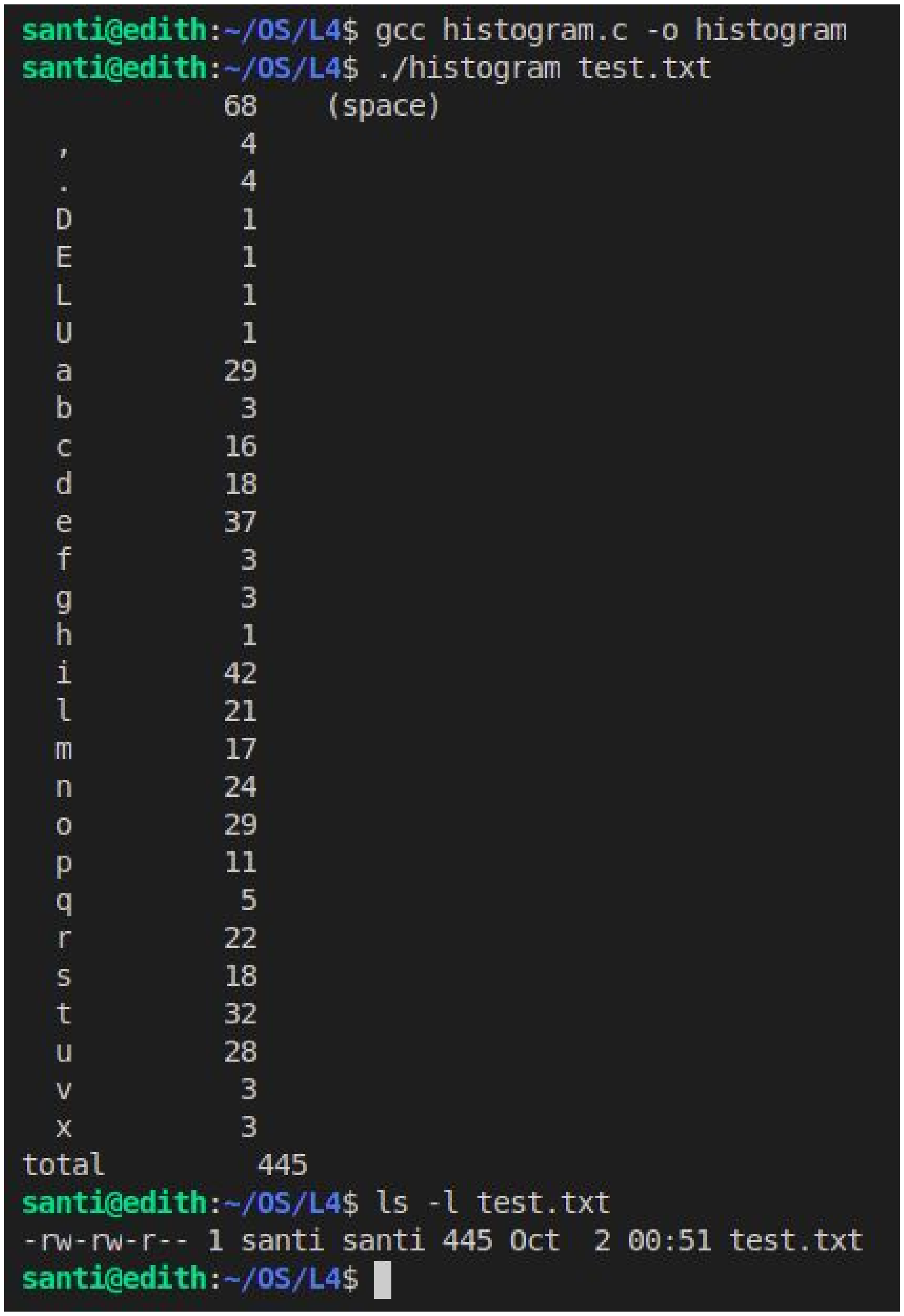


```
my-command-shell:~@ cd OS/L4
my-command-shell:~/OS/L4@ ls
'Fork Assignment-3.pdf'   max_no_of_proc_1.c   max_no_of_proc_2.c   myshell   orpha
n.c   zombie.c
my-command-shell:~/OS/L4@ !20
history: you have executed only 15 commands so far including '!20'
my-command-shell:~/OS/L4@ !10
history of commands executed (recent first) :
    16  !10
    15  !20
    14  ls
    13  cd OS/L4
    12  cowsay -f duck quack!
    11  !15
    10  !6
     9  whatis cp
     8  cd ~
     7  factor 10
my-command-shell:~/OS/L4@ exit
santi@edith:~/OS/L4/myshell$ 
```

5) Develop a multiprocessing version of Histogram generator to count the occurrence of various characters in a given text.

Explanation - Please refer to the comments in the histogram.c file

Output

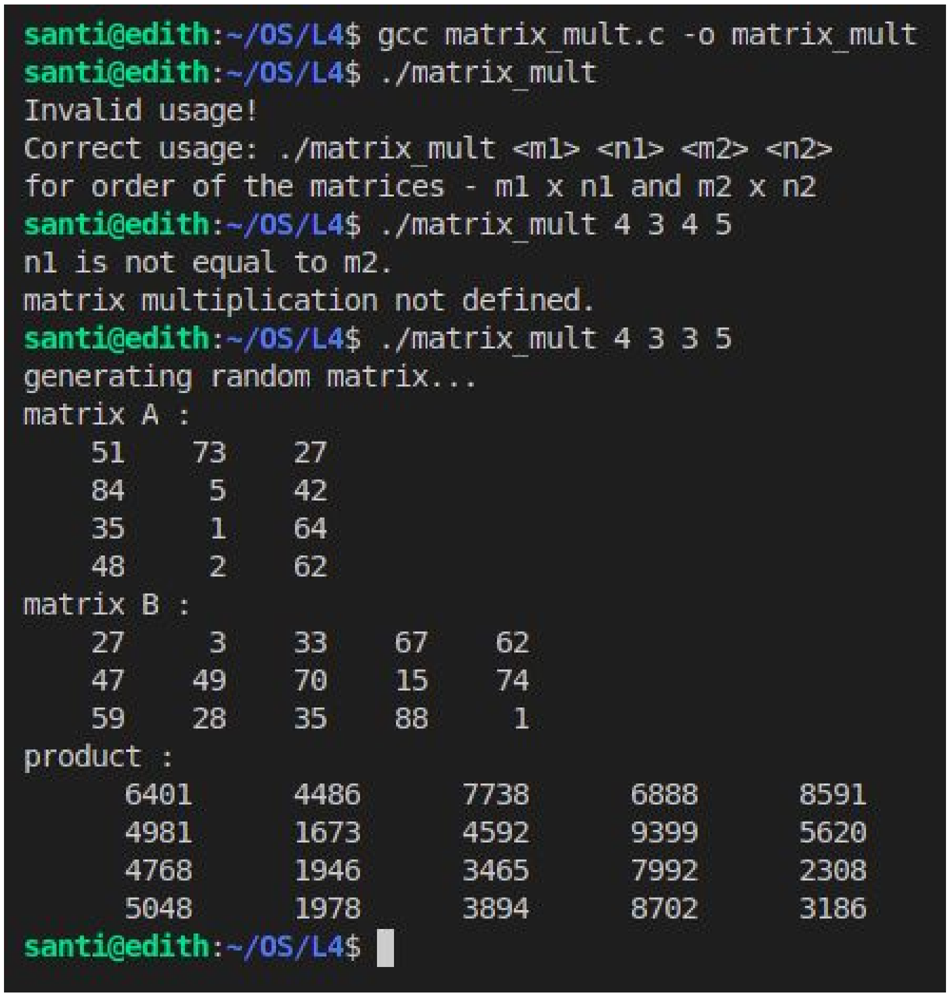


```
santi@edith:~/OS/L4$ gcc histogram.c -o histogram
santi@edith:~/OS/L4$ ./histogram test.txt
                68     (space)
  ,              4
  .              4
  D              1
  E              1
  L              1
  U              1
  a             29
  b              3
  c             16
  d             18
  e             37
  f              3
  g              3
  h              1
  i             42
  l             21
  m             17
  n             24
  o             29
  p             11
  q              5
  r             22
  s             18
  t             32
  u             28
  v              3
  x              3
total            445
santi@edith:~/OS/L4$ ls -l test.txt
-rw-rw-r-- 1 santi santi 445 Oct  2 00:51 test.txt
santi@edith:~/OS/L4$ 
```

6) Develop a multiprocessing version of matrix multiplication. Say for a result 3*3 matrix, the most efficient form of parallelization can be 9 processes, each of which computes the net resultant value of a row (matrix1) multiplied by column (matrix2). For programmers convenience you can start with 4 processes, but as I said each result value can be computed parallel independent of the other processes in execution.

Explanation - Please refer to the comments in the matrix_mult.c file

Output



```
santi@edith:~/OS/L4$ gcc matrix_mult.c -o matrix_mult
santi@edith:~/OS/L4$ ./matrix_mult
Invalid usage!
Correct usage: ./matrix_mult <m1> <n1> <m2> <n2>
for order of the matrices - m1 x n1 and m2 x n2
santi@edith:~/OS/L4$ ./matrix_mult 4 3 4 5
n1 is not equal to m2.
matrix multiplication not defined.
santi@edith:~/OS/L4$ ./matrix_mult 4 3 3 5
generating random matrix...
matrix A :
    51    73    27
    84     5    42
    35     1    64
    48     2    62
matrix B :
    27     3    33    67    62
    47    49    70    15    74
    59    28    35    88     1
product :
      6401      4486      7738      6888      8591
      4981      1673      4592      9399      5620
      4768      1946      3465      7992      2308
      5048      1978      3894      8702      3186
santi@edith:~/OS/L4$ 
```

Non Mandatory (Extra Credits)

7) Develop a parallelized application to check for if a user input square matrix is a magic square or not. No of processes again can be optimal as w.r.t to matrix exercise above.

Explanation:

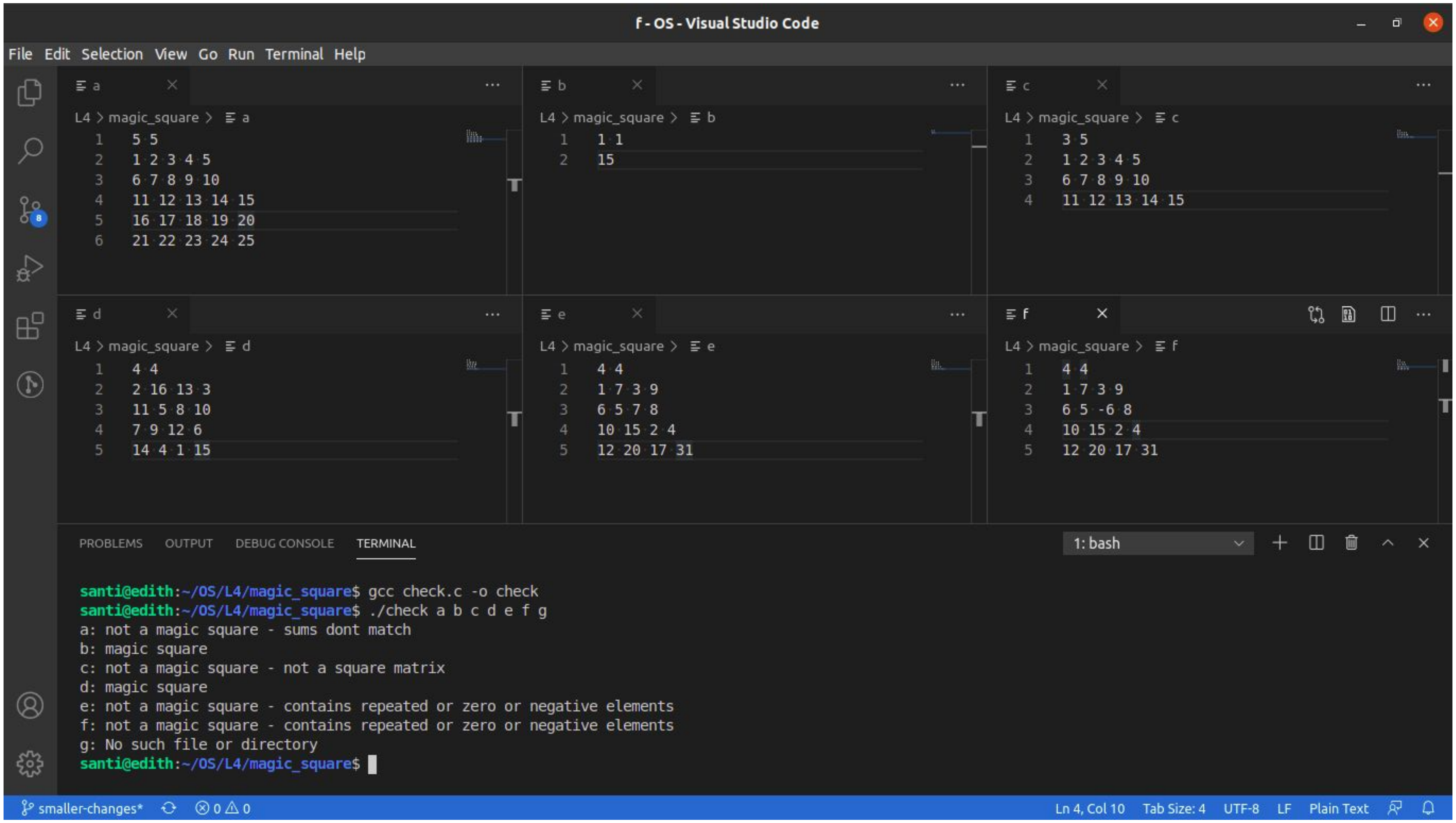
The `main()` function does the following:

- Receives filenames from command line arguments, report error if command line arguments are not found.
- For each file:
  - run `check_magic_square(filename)` and store its return value in the flag variable.
  - Print message according to the flag value
- Return.

The `check_magic_square(filename)` function does the following:

- Open the given file, return -3 if file is not found.
- Read file data. Return -2 if row!=col
- Create an shm array, read data from file and it in the created shm array.
- Check if entries in the matrix are unique and all the elements are >0
  - If no, return -1
  - Else continue
- Create a shm array for sums and parallelly calculate row, diagonal, column sums and store it in the array.
- Set flag=1
- Check if any element in the sum array is different. If so, change flag to 0.
- Free the shm array created and return flag.

Output - screenshot is given on the next page


f - OS - Visual Studio Code
File Edit Selection View Go Run Terminal Help
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1: bash
smaller-changes* 0 0
Ln 4, Col 10 Tab Size: 4 UTF-8 LF Plain Text

L4 > magic_square > a

```
5 5
1 2 3 4 5
6 7 8 9 10
11 12 13 14 15
16 17 18 19 20
21 22 23 24 25
```

L4 > magic_square > b

```
1 1
15
```

L4 > magic_square > c

```
3 5
1 2 3 4 5
6 7 8 9 10
11 12 13 14 15
```

L4 > magic_square > d

```
4 4
2 16 13 3
11 5 8 10
7 9 12 6
14 4 1 15
```

L4 > magic_square > e

```
4 4
1 7 3 9
6 5 7 8
10 15 2 4
12 20 17 31
```

L4 > magic_square > f

```
4 4
1 7 3 9
6 5 -6 8
10 15 2 4
12 20 17 31
```

```
santi@edith:~/OS/L4/magic_square$ gcc check.c -o check
santi@edith:~/OS/L4/magic_square$ ./check a b c d e f g
a: not a magic square - sums dont match
b: magic square
c: not a magic square - not a square matrix
d: magic square
e: not a magic square - contains repeated or zero or negative elements
f: not a magic square - contains repeated or zero or negative elements
g: No such file or directory
santi@edith:~/OS/L4/magic_square$
```

8) Extend the above to also support magic square generation (u can take as input the order of the matrix..refer the net for algorithms for odd and even version...)

Explanation

There are 3 cases in magic square generation:

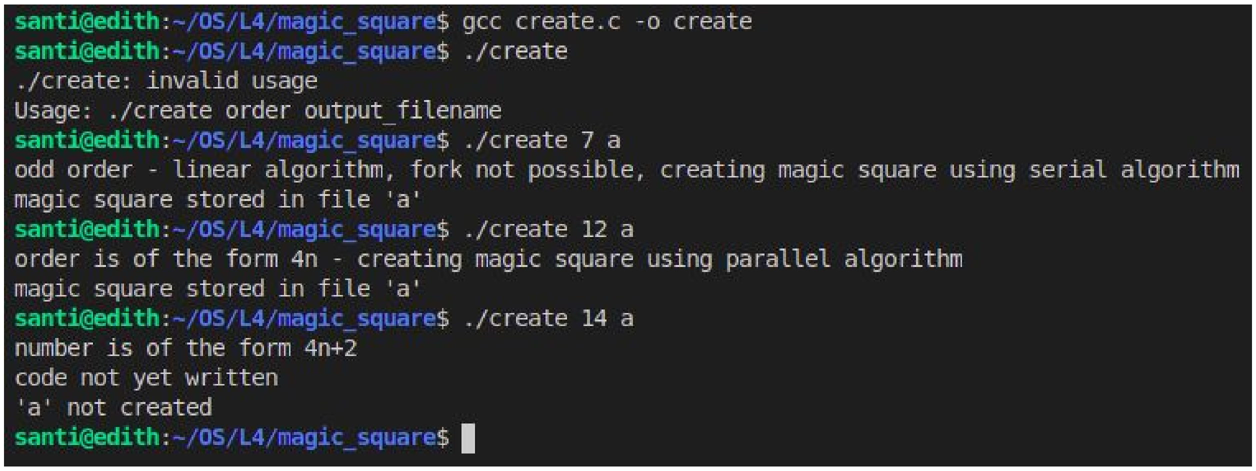
1) Odd order
2) Doubly even (of the form 4n)
3) Singly even (of the form 4n+2)

Odd order has a linear algorithm in which every step depends on the previous step so it has been implemented in a serial algorithm.

Doubly even has 5 different processes which are independent and can go on parallelly so it has been implemented using 5 `fork()` calls parallelly.

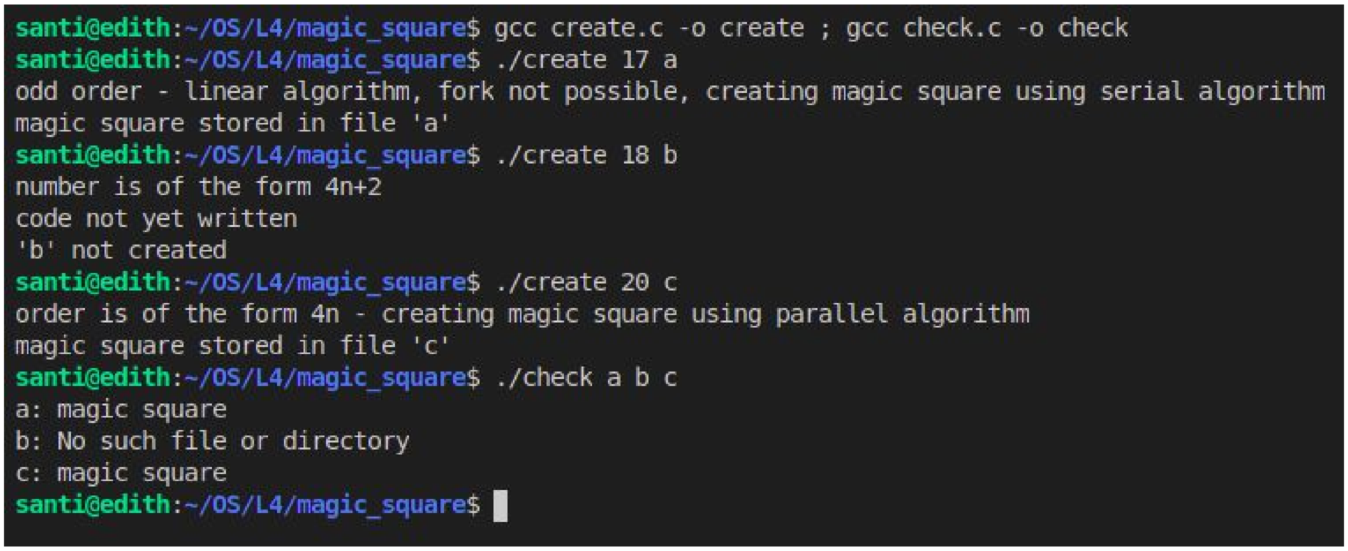
Singly even - I could not find any legitimate algorithm anywhere and since this is an extra credit question and there is a submission deadline, I have decided to skip this for now. I will submit this part later.

Output

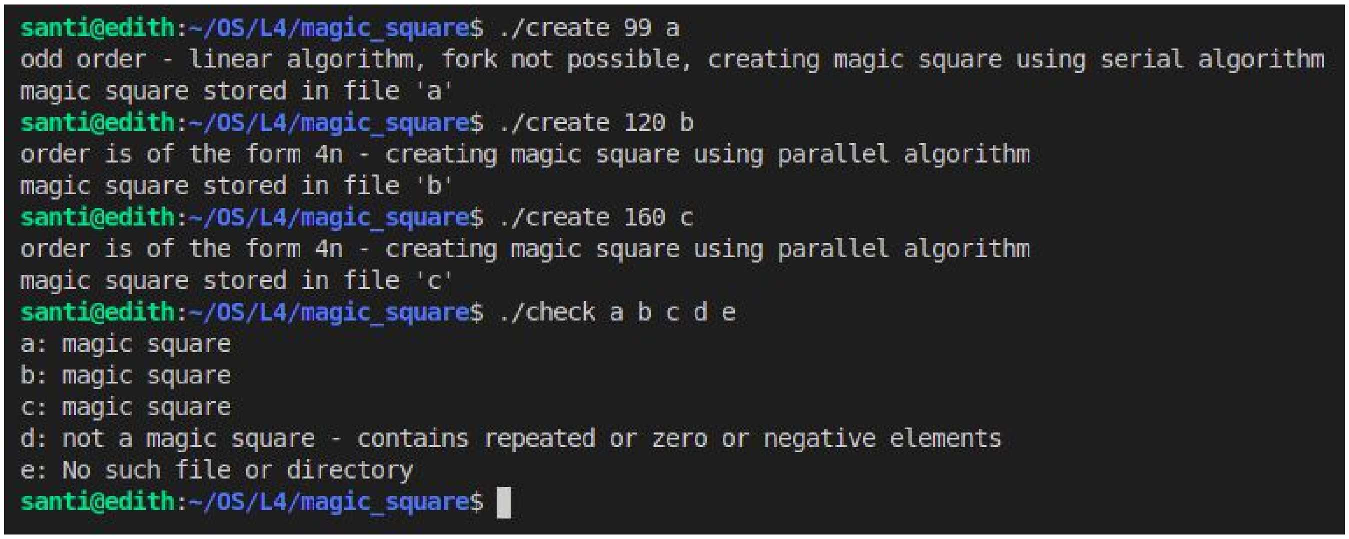


```
santi@edith:~/OS/L4/magic_square$ gcc create.c -o create
santi@edith:~/OS/L4/magic_square$ ./create
./create: invalid usage
Usage: ./create order output_filename
santi@edith:~/OS/L4/magic_square$ ./create 7 a
odd order - linear algorithm, fork not possible, creating magic square using serial algorithm
magic square stored in file 'a'
santi@edith:~/OS/L4/magic_square$ ./create 12 a
order is of the form 4n - creating magic square using parallel algorithm
magic square stored in file 'a'
santi@edith:~/OS/L4/magic_square$ ./create 14 a
number is of the form 4n+2
code not yet written
'a' not created
santi@edith:~/OS/L4/magic_square$
```

Checking output of `./create` using `./check`



```
santi@edith:~/OS/L4/magic_square$ gcc create.c -o create ; gcc check.c -o check
santi@edith:~/OS/L4/magic_square$ ./create 17 a
odd order - linear algorithm, fork not possible, creating magic square using serial algorithm
magic square stored in file 'a'
santi@edith:~/OS/L4/magic_square$ ./create 18 b
number is of the form 4n+2
code not yet written
'b' not created
santi@edith:~/OS/L4/magic_square$ ./create 20 c
order is of the form 4n - creating magic square using parallel algorithm
magic square stored in file 'c'
santi@edith:~/OS/L4/magic_square$ ./check a b c
a: magic square
b: No such file or directory
c: magic square
santi@edith:~/OS/L4/magic_square$
```



```
santi@edith:~/OS/L4/magic_square$ ./create 99 a
odd order - linear algorithm, fork not possible, creating magic square using serial algorithm
magic square stored in file 'a'
santi@edith:~/OS/L4/magic_square$ ./create 120 b
order is of the form 4n - creating magic square using parallel algorithm
magic square stored in file 'b'
santi@edith:~/OS/L4/magic_square$ ./create 160 c
order is of the form 4n - creating magic square using parallel algorithm
magic square stored in file 'c'
santi@edith:~/OS/L4/magic_square$ ./check a b c d e
a: magic square
b: magic square
c: magic square
d: not a magic square - contains repeated or zero or negative elements
e: No such file or directory
santi@edith:~/OS/L4/magic_square$
```