Assignment 4 - Fork

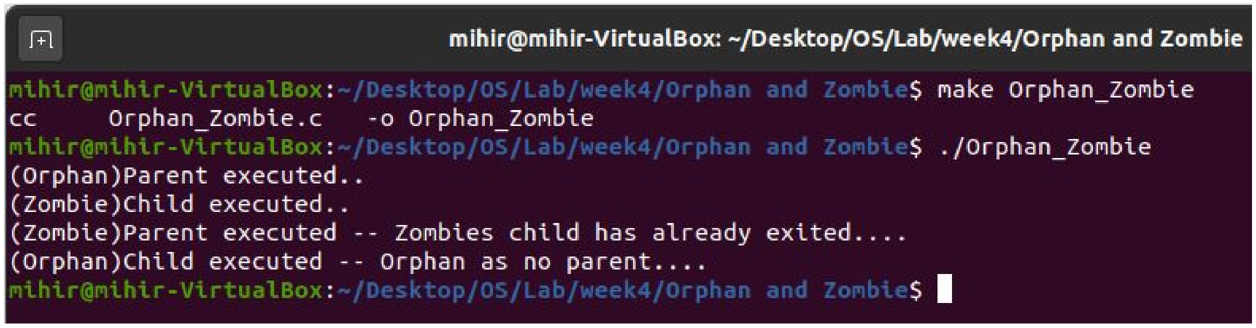
(1) Test drive a C program that creates Orphan and Zombie Processes.

Code:

```
#include<stdio.h>
#include<unistd.h>
#include<string.h>
#include<stdlib.h>
#include<sys/wait.h>

int main()
{
   int pid = fork();

   if(pid == 0)
   {
         int pido = fork();

         if(pido > 0)    // Parent
         {
                 printf("(Orphan)Parent executed..\n");
                 return 0;
         }

         else if(pido == 0)    // Child
         {
                 sleep(2.5);    // Wait 2.5

                 printf("(Orphan)Child executed -- Orphan as no parent....\n");
                 return 0;
         }

         printf("\n");
   }

   else if(pid > 0)
```

```
    {
            wait(NULL);

            int pidz = fork();

            if(pidz == 0)    // Child
            {
                    printf("(Zombie)Child executed..\n");
                    exit(0);
                    return 0;
            }

            else if(pidz > 0)    // Parent
            {
                    sleep(2.5);    // Wait 2.5     // Now this parent is executed but all of its
children have executed already and exited

                    printf("(Zombie)Parent executed -- Zombies child has already
exited....\n");
                    return 0;
            }

            printf("\n");
            return 0;
    }

}
```

Output ss:



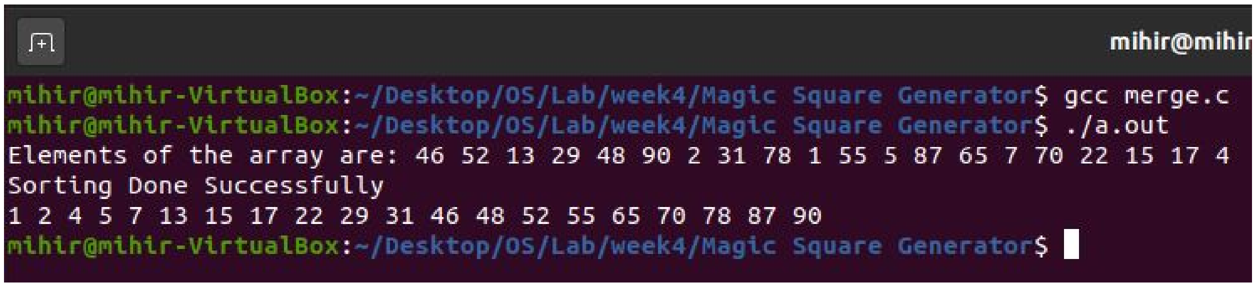
```
mihir@mihir-VirtualBox: ~/Desktop/OS/Lab/week4/Orphan and Zombie
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week4/Orphan and Zombie$ make Orphan_Zombie
cc     Orphan_Zombie.c    -o Orphan_Zombie
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week4/Orphan and Zombie$ ./Orphan_Zombie
(Orphan)Parent executed..
(Zombie)Child executed..
(Zombie)Parent executed -- Zombies child has already exited....
(Orphan)Child executed -- Orphan as no parent....
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week4/Orphan and Zombie$
```

(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]

Logic: Recursively make two child processes, one for the left half, one of the right half. If the number of elements in the array for a process is less than 5, perform an Insertion Sort. The parent of the two children then merges the result and returns back to its parent and so on.

Code:
```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

void insertionSort(int arr[], int n);
void merge(int a[], int l1, int h1, int h2);

void mergeSort(int a[], int l, int h)
{
    int i, len=(h-l+1);

    // Using insertion sort for small sized array
    if (len<=5)
    {
        insertionSort(a+l, len);
        return;
    }

    pid_t lpid,rpid;
    lpid = vfork();
    if (lpid<0)
    {
        // Lchild process not created
        perror("Left Child Proc. not created\n");
        _exit(-1);
    }
    else if (lpid==0)
    {
            mergeSort(a,l,l+len/2-1);
            _exit(0);
    }
    else
    {
            rpid = vfork();
            if (rpid<0)
            {
                    // Rchild process not created
                    perror("Right Child Proc. not created\n");
                    _exit(-1);
            }
            else if(rpid==0)
            {
                    mergeSort(a,l+len/2,h);
                    _exit(0);
            }
    }

    int status;

    // Wait for child processes to finish
    waitpid(lpid, &status, 0);
    waitpid(rpid, &status, 0);

    // Merge the sorted subarrays
    merge(a, l, l+len/2-1, h);

}

/* Function to sort an array using insertion sort*/
void insertionSort(int arr[], int n)
{
int i, key, j;
for (i = 1; i < n; i++)
{
    key = arr[i];
    j = i-1;

    /* Move elements of arr[0..i-1], that are greater than key, to one position ahead
  of their current position */
    while (j >= 0 && arr[j] > key)
    {
            arr[j+1] = arr[j];
            j = j-1;
    }
    arr[j+1] = key;
}
}

// Method to merge sorted subarrays
void merge(int a[], int l1, int h1, int h2)
{
    // We can directly copy the sorted elements in the final array, no need for a temporary
    // sorted array.
    int count=h2-l1+1;
    int sorted[count];
    int i=l1, k=h1+1, m=0;
    while (i<=h1 && k<=h2)
    {
            if (a[i]<a[k])
                    sorted[m++]=a[i++];
            else if (a[k]<a[i])
                    sorted[m++]=a[k++];
            else if (a[i]==a[k])
            {
                    sorted[m++]=a[i++];
                    sorted[m++]=a[k++];
            }
    }

    while (i<=h1)
            sorted[m++]=a[i++];

    while (k<=h2)
            sorted[m++]=a[k++];
    int arr_count = l1;
    for (i=0; i<count; i++,l1++)
            a[l1] = sorted[i];

    for (int i = 0; i <= h2; i++)
            printf("%d ", a[i]);
}

// To check if array is actually sorted or not
void isSorted(int arr[], int len)
{
    if (len==1)
    {
            printf("\nSorting Done Successfully\n");
            return;
    }

    int i;
    for (i=1; i<len; i++)
    {
            if (arr[i]<arr[i-1])
            {
                    printf("\nSorting Not Done\n");
                    return;
            }
    }
    printf("\nSorting Done Successfully\n");
    return;
}

// Driver code
int main()
{
    int length = 20;
    int shm_array[] = {46, 52, 13, 29, 48, 90, 2, 31, 78, 1, 55, 5, 87, 65, 7, 70, 22, 15, 17,
4};

    printf("Elements of the array are: ");
    for (int i = 0; i < length; ++i)
    {
        printf("%d ", shm_array[i]);
    }

    // Sort the created array
    mergeSort(shm_array, 0, length-1);

    // Check if array is sorted or not
    isSorted(shm_array, length);

    for (int i = 0; i < length; i++)
        printf("%d ", shm_array[i]);

    printf("\n");

    return 0;

}
```

Output ss:



```
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week4/Magic Square Generator$ gcc merge.c
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week4/Magic Square Generator$ ./a.out
Elements of the array are: 46 52 13 29 48 90 2 31 78 1 55 5 87 65 7 70 22 15 17 4
Sorting Done Successfully
1 2 4 5 7 13 15 17 22 29 31 46 48 52 55 65 70 78 87 90
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week4/Magic Square Generator$
```

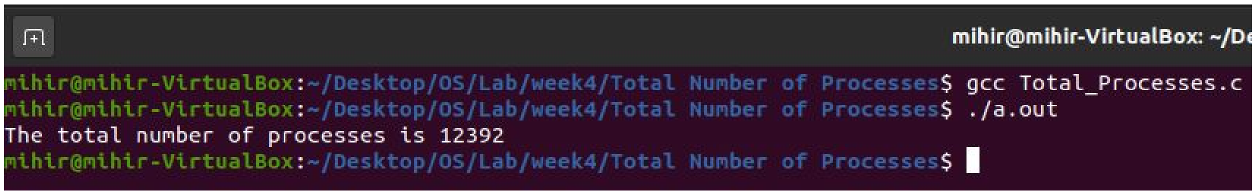
(3) Develop a C program to count the maximum number of processes that can be created using fork call.

Logic: The basic logic is to keep forking until we cannot fork anymore i.e. keep forking until fork returns -1. To do this, apply a while(1) loop and keep forking within it. Keep checking for the status of pid. If pid == 0 i.e. control is with a child process, kill it because we only want to count the number of processes. If pid < 0 then print count+1 (+1 because parent process will also be counted) and break out of the loop.

Code:

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

int main()
{
   int c = 0;

   while(1)
   {
         int pid = fork();

         if (pid < 0)
         {
                 printf("The total number of processes is %d\n", c+1);
                 break;
         }

         if (pid == 0)
                 exit(0);

         c++;
   }

   return 0;

}
```

Output ss:



```
mihir@mihir-VirtualBox: ~/De
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week4/Total Number of Processes$ gcc Total_Processes.c
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week4/Total Number of Processes$ ./a.out
The total number of processes is 12392
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week4/Total Number of Processes$
```

(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.

Logic: The logic behind implementing the system commands or user binaries is the use of **execvp() function**. We ask the user to input a command and store it in a character array. We separate the main command with its options using a GetArgs function which takes in a string and splits the string on finding white spaces. So, if the user gives ls -l it will be seen as “ls” and “-l” and that’s how it will be passed to the execvp() function. The execvp function is called as part of a child process so whichever command the user enters, the child takes up the image of that command and executes it. If execvp returns -1 i.e. the command entered by the user is invalid, we throw an error. We allow the user to keep entering as many commands as he wants until he decides to stop. In order to stop, the user has to type “exit”.

In order to implement the history feature, we store the commands entered by the user in an array upto a certain limit (I put the limit as 10). Whenever the user types something starting with an “!” the system knows that the user wants to know the history of the commands. So, if the user types !5 the 5 after the “!” is converted to an integer and passed to the DisplayHistory() function which loops through the stored commands array and displays the command history. If the user enters a number which is greater than the total number of commands in the command history array then the entire history is displayed.

Code:

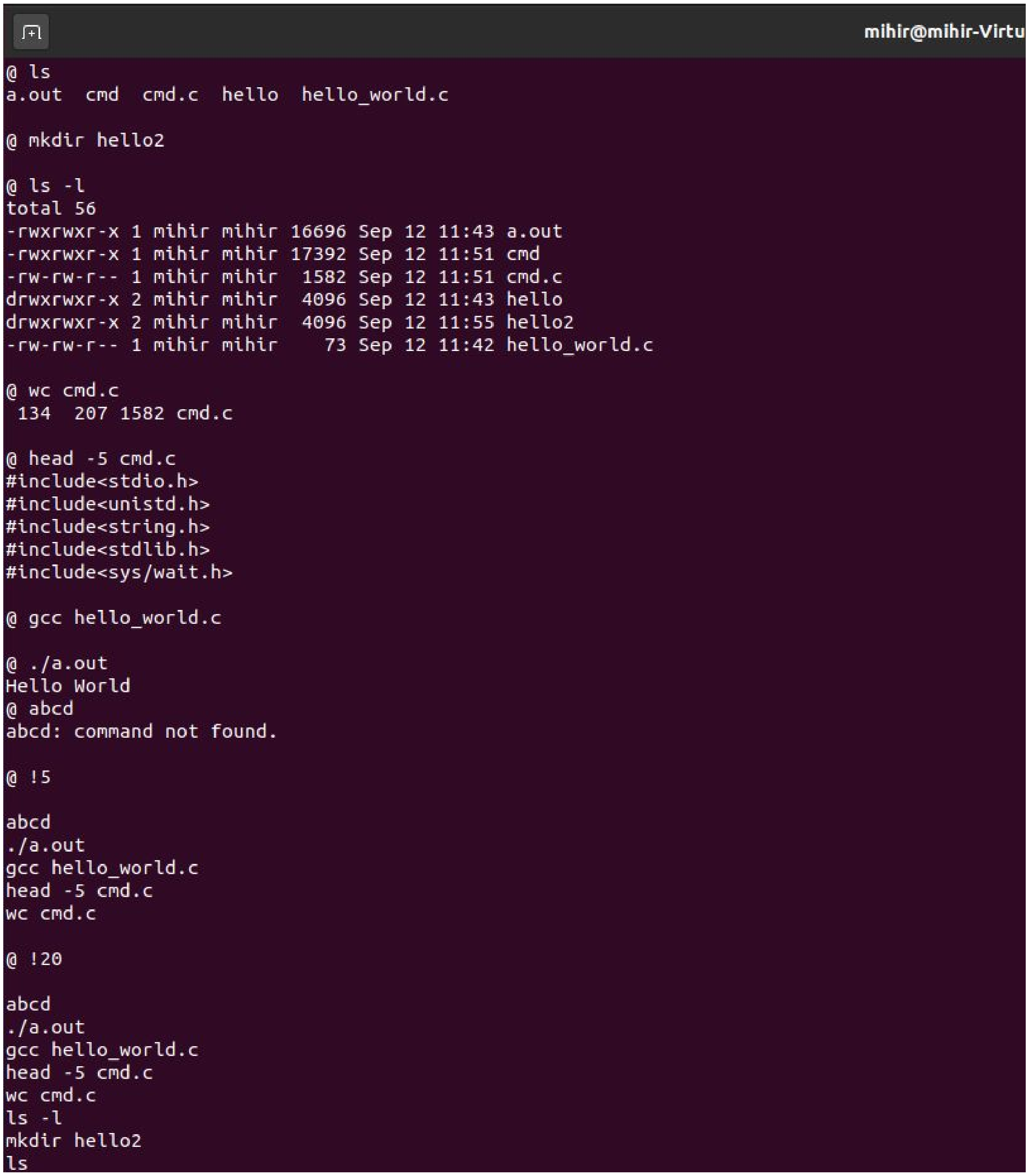
```
#include<stdio.h>
#include<unistd.h>
#include<string.h>
#include<stdlib.h>
#include<sys/wait.h>
#define maxcmd 10

char *argp[100];
int arglen;
char buf[100][100];
int buflen;

char PrevCmd[maxcmd][500];
int cmdno;

void GetArgs(char c[])
{
   int i=0, eol = 0;
   arglen = 0;
   buflen = 0;

   while(eol == 0)
   {
         if(c[i] == '\0')
         {
                  eol = 1;

                  buf[arglen][buflen] = '\0';

                  argp[arglen] = buf[arglen];
                  arglen++;

                  buflen = 0;
         }

         else
```

```
            {
                    if(c[i] == ' ')
                    {
                            buf[arglen][buflen] = '\0';

                            argp[arglen] = buf[arglen];
                            arglen++;

                            buflen = 0;
                    }

                    else
                    {
                            buf[arglen][buflen] = c[i];
                            buflen++;
                    }
            }

            i++;
      }

      argp[arglen] = NULL;
}

void DisplayHistory(int h)
{
      printf("\n");

      for(int i=0;i<h && i<maxcmd && cmdno > i;i++)
      {
            int j = (cmdno-1-i)%maxcmd;
            printf("%s\n", PrevCmd[j]);
      }

      printf("\n");
}

int main()
{
```

```
char cmd[500];
cmdno = 0;
while(1)
{
	printf("@ ");
	scanf(" %[^\n]", cmd);

	if(strcmp(cmd, "exit") == 0)
	{
		return 0;
	}

	else if(cmd[0] == '!')
	{
		int h;
		if (!cmd[2])
			h = (int)cmd[1] - 48;
		else
		{
			int x = (int)cmd[1] - 48;
			int y = (int)cmd[2] - 48;
			h = 10*x + y;
		}

		DisplayHistory(h);
	}

	else
	{
		int pid = vfork();
		if(pid == 0)
		{
			strcpy(PrevCmd[cmdno%maxcmd], cmd);
			cmdno++;

			GetArgs(cmd);

			if(execvp(argp[0], argp) == -1)
			{
```

```
                        printf("%s: command not found.\n", argp[0]);
                    }
                    exit(0);
                }
                else
                {
                    wait(NULL);
                    printf("\n");
                }
            }
        }
}
```

Output ss:



```
@ ls
a.out  cmd  cmd.c  hello  hello_world.c

@ mkdir hello2

@ ls -l
total 56
-rwxrwxr-x 1 mihir mihir 16696 Sep 12 11:43 a.out
-rwxrwxr-x 1 mihir mihir 17392 Sep 12 11:51 cmd
-rw-rw-r-- 1 mihir mihir  1582 Sep 12 11:51 cmd.c
drwxrwxr-x 2 mihir mihir  4096 Sep 12 11:43 hello
drwxrwxr-x 2 mihir mihir  4096 Sep 12 11:55 hello2
-rw-rw-r-- 1 mihir mihir    73 Sep 12 11:42 hello_world.c

@ wc cmd.c
 134  207 1582 cmd.c

@ head -5 cmd.c
#include<stdio.h>
#include<unistd.h>
#include<string.h>
#include<stdlib.h>
#include<sys/wait.h>

@ gcc hello_world.c

@ ./a.out
Hello World
@ abcd
abcd: command not found.

@ !5

abcd
./a.out
gcc hello_world.c
head -5 cmd.c
wc cmd.c

@ !20

abcd
./a.out
gcc hello_world.c
head -5 cmd.c
wc cmd.c
ls -l
mkdir hello2
ls
```

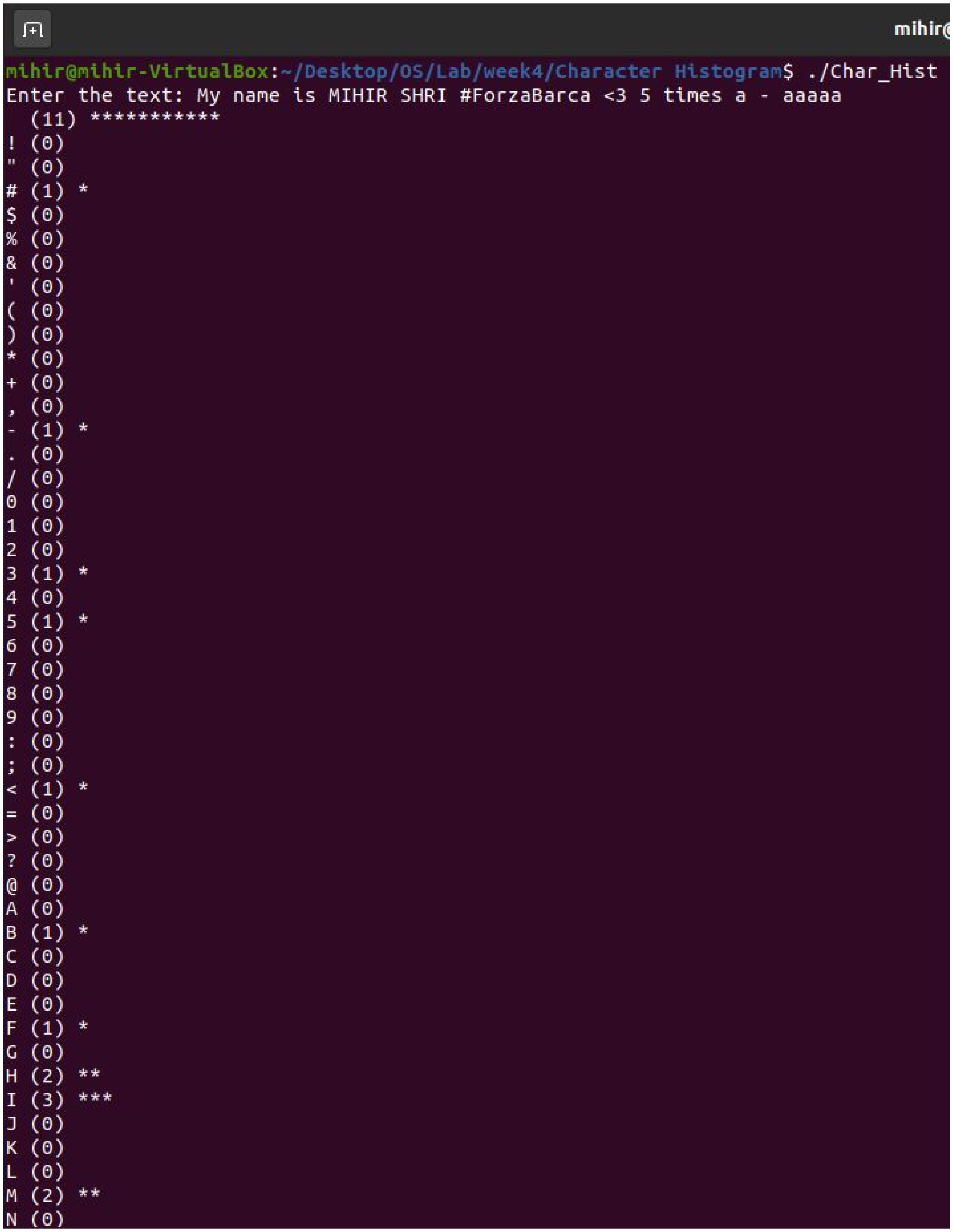
(5) Develop a multiprocessing version of Histogram generator to count the occurrence of various characters in a given text.

Logic: We implement the Character Histogram using **Hashing.** We use **vfork()** to create parent and child processes. The child process loops through the entire string entered by the user and increases the value of a character in the Hash Table as soon as it is encountered. **vfork()** is used so that the parent process can access the Hash Table created by the child.
The parent process loops through the Hash Table and prints each character along with its count.

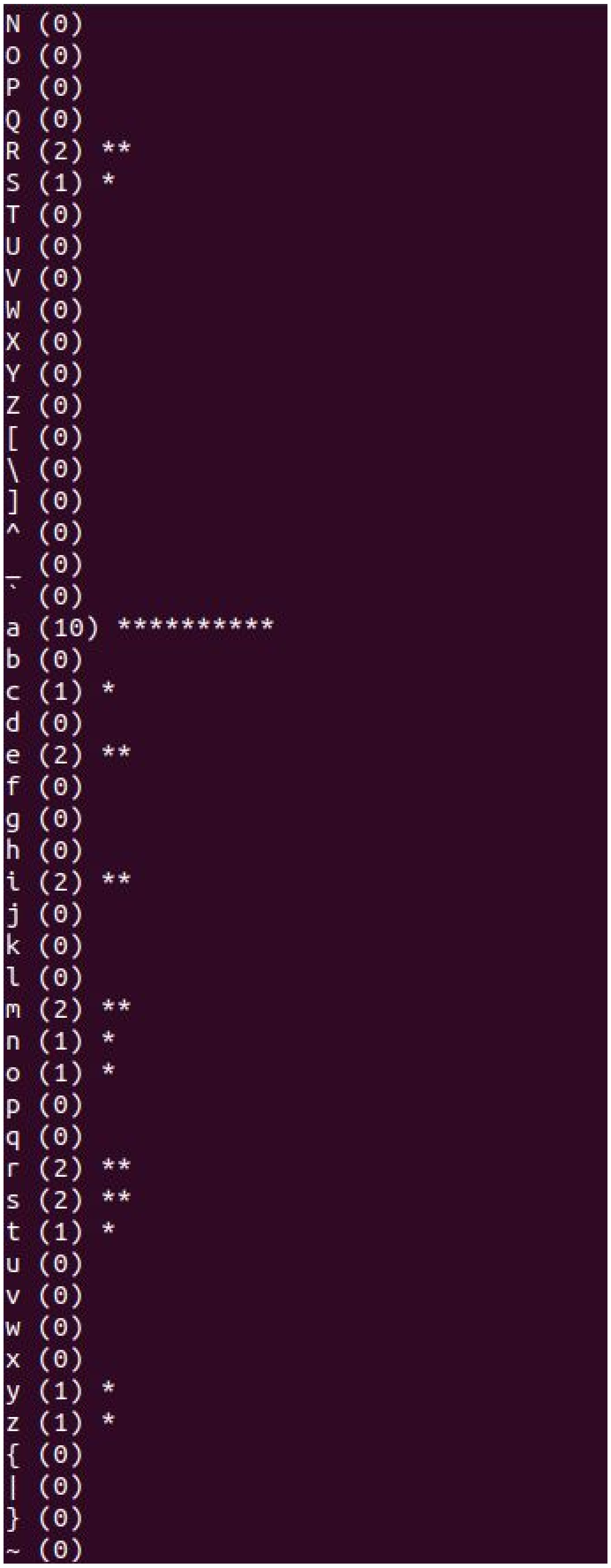
Code:

```
#include<stdio.h>
#include<unistd.h>
#include<string.h>
#include<sys/wait.h>
#include<stdlib.h>

#define totchars 128

int main()
{
   char text[500];

   while(1)
   {
         printf("Enter the text: ");
         gets(text);

         int freq[totchars] = {0};

         int pid = vfork();

         if(pid == 0)
         {
                for(int i=0;i<strlen(text);i++)
                {
                        freq[(int)text[i]]++;
                }
                exit(0);
        }

        else if (pid > 0)
        {
                wait(NULL);

                for(int i=32;i<totchars-1;i++)
                {
                        printf("%c (%d) ", (char)i, freq[i]);
                        for(int j=0;j<freq[i];j++) printf("*");
                        printf("\n");
                }

                printf("\n\n");

                int exit = 0;
                printf("Do you want to exit: ");
                scanf("%d", &exit);

                if(exit == 1)
                {
                        return 0;
                }
        }
    }
}
```

Output ss:



```
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week4/Character Histogram$ ./Char_Hist
Enter the text: My name is MIHIR SHRI #ForzaBarca <3 5 times a - aaaaa
  (11) ***********
! (0)
" (0)
# (1) *
$ (0)
% (0)
& (0)
' (0)
( (0)
) (0)
* (0)
+ (0)
, (0)
- (1) *
. (0)
/ (0)
0 (0)
1 (0)
2 (0)
3 (1) *
4 (0)
5 (1) *
6 (0)
7 (0)
8 (0)
9 (0)
: (0)
; (0)
< (1) *
= (0)
> (0)
? (0)
@ (0)
A (0)
B (1) *
C (0)
D (0)
E (0)
F (1) *
G (0)
H (2) **
I (3) ***
J (0)
K (0)
L (0)
M (2) **
N (0)
```



```
N (0)
O (0)
P (0)
Q (0)
R (2) **
S (1) *
T (0)
U (0)
V (0)
W (0)
X (0)
Y (0)
Z (0)
[ (0)
\ (0)
] (0)
^ (0)
_ (0)
` (0)
a (10) **********
b (0)
c (1) *
d (0)
e (2) **
f (0)
g (0)
h (0)
i (2) **
j (0)
k (0)
l (0)
m (2) **
n (1) *
o (1) *
p (0)
q (0)
r (2) **
s (2) **
t (1) *
u (0)
v (0)
w (0)
x (0)
y (1) *
z (1) *
{ (0)
| (0)
} (0)
~ (0)
```

(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. Non Mandatory (Extra Credits).

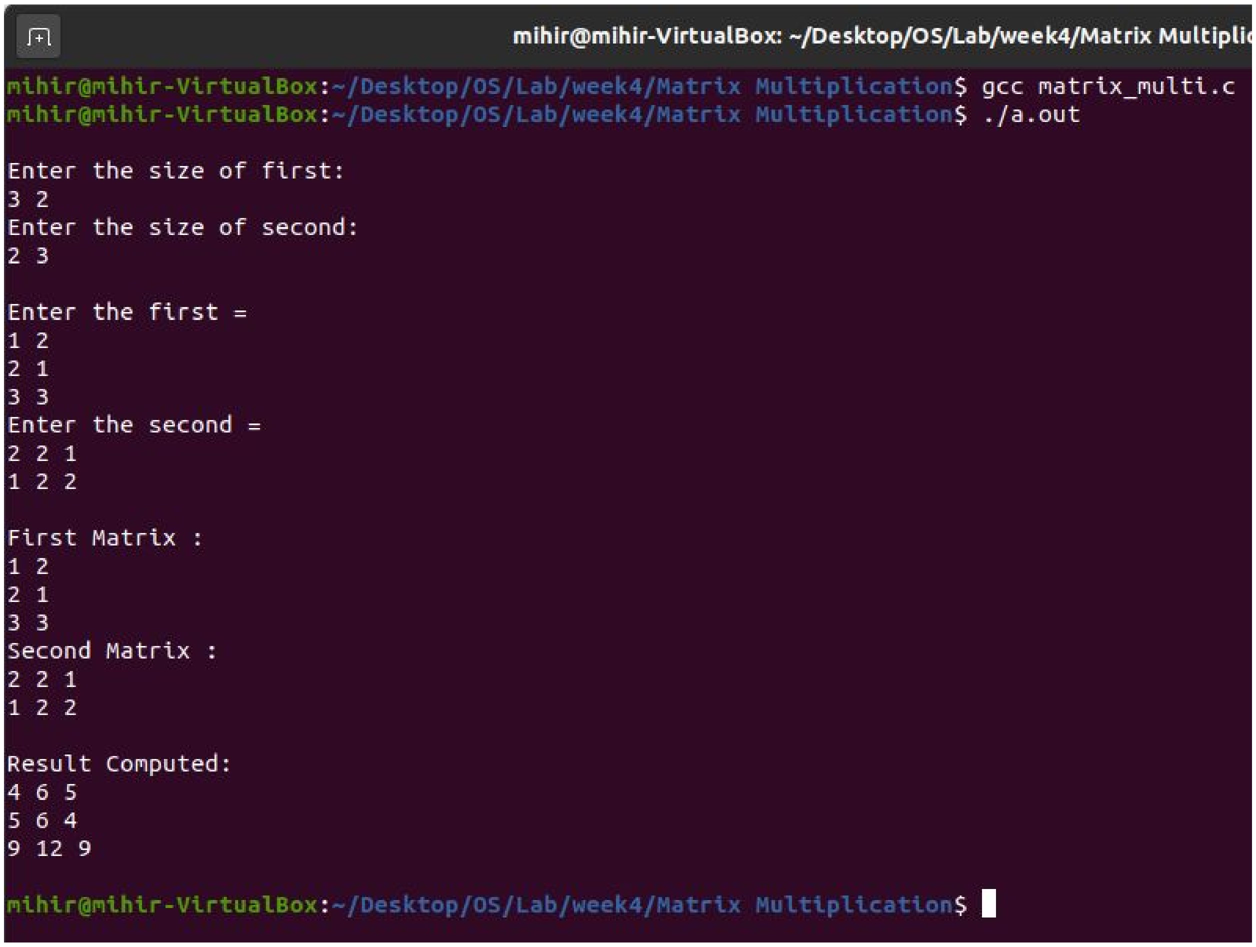
Logic: As mentioned in the question, the idea is to take an array of PIDs. Each value in this array corresponds to the pid of a process which will be created for computing the net resultant value of a row (matrix1) multiplied by column (matrix2). The matrix as stated in the question, each multiplication is parallelized in the most efficient way using vfork() where the data is shared across all the process and the overall output is accumulated and displayed in the end.

Code:

```
#include <stdio.h>
#include <sys/wait.h>
#include <unistd.h>
#include <time.h>
#include <stdlib.h>
int r1, c1, r2, c2;
void get_input(int a, int b, int array[][b]) {          //take input from the arrays
        for (int i = 0; i < a; i++)
                for (int j = 0; j < b; j++) {
                scanf("%d", & array[i][j]);
                }
}

void display(int a, int b, int array[][b]) {            //display a 2-D array
        for (int i = 0; i < a; i++) {
                for (int j = 0; j < b; j++) {
                printf("%d ", array[i][j]);
                }

                printf("\n");
        }
}
```

```
int matmul(int a, int b, int a1[][c1], int a2[][c2])  //basic multiplication (RC)
{
        int sum = 0;
        for (int i = 0; i < r2; i++)
                sum += a1[a][i] * a2[i][b];
        return sum;
}

int main() {
        int status;

        printf("\nEnter the size of first:\n");
        scanf("%d %d", & r1, & c1);
        printf("Enter the size of second:\n");
        scanf("%d %d", & r2, & c2);

        int a[r1][c1];
        int b[r2][c2];

        printf("\nEnter the first =\n");
        get_input(r1, c1, a);
        printf("Enter the second =\n");
        get_input(r2, c2, b);

        printf("\nFirst Matrix :\n");
        display(r1, c1, a);
        printf("Second Matrix :\n");
        display(r2, c2, b);

        int c[r1][c2];
        printf("\nResult Computed:\n");
        pid_t pid[r1 * c2];
        int index = 0;
        int sum1, sum2;

        for (int i = 0; i < r1; i++) {

                for (int j = 0; j < c2; j += 2) {
                pid[index] = vfork(); //use of vfork()
                if (pid[index++] == 0) {
                        sum1 = matmul(i, j, a, b);
                        c[i][j] = sum1;



                        _exit(0);
                } else {
                        if (j + 1 < c2) {
                                sum2 = matmul(i, j + 1, a, b);
                                c[i][j + 1] = sum2;



                        }
                }
                }
        }

        waitpid(-1, & status, 0);
        display(r1, c2, c);

        printf("\n");

        return 0;
}
```

Output ss:


mihir@mihir-VirtualBox: ~/Desktop/OS/Lab/week4/Matrix Multiplic

```
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week4/Matrix Multiplication$ gcc matrix_multi.c
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week4/Matrix Multiplication$ ./a.out

Enter the size of first:
3 2
Enter the size of second:
2 3

Enter the first =
1 2
2 1
3 3
Enter the second =
2 2 1
1 2 2

First Matrix :
1 2
2 1
3 3
Second Matrix :
2 2 1
1 2 2

Result Computed:
4 6 5
5 6 4
9 12 9

mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week4/Matrix Multiplication$ 
```

(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.

Logic: We create 4 processes and assign each of them with a specific task.

Process 1 - calculates the sum of each row and does this for all the rows.
Process 2 - calculates the sum of each column and does this for all the columns.
Process 3 - calculates the sum of each diagonal and does this for both the diagonals.
Process 4 - checks if all the elements in the magic square are unique.

Code:

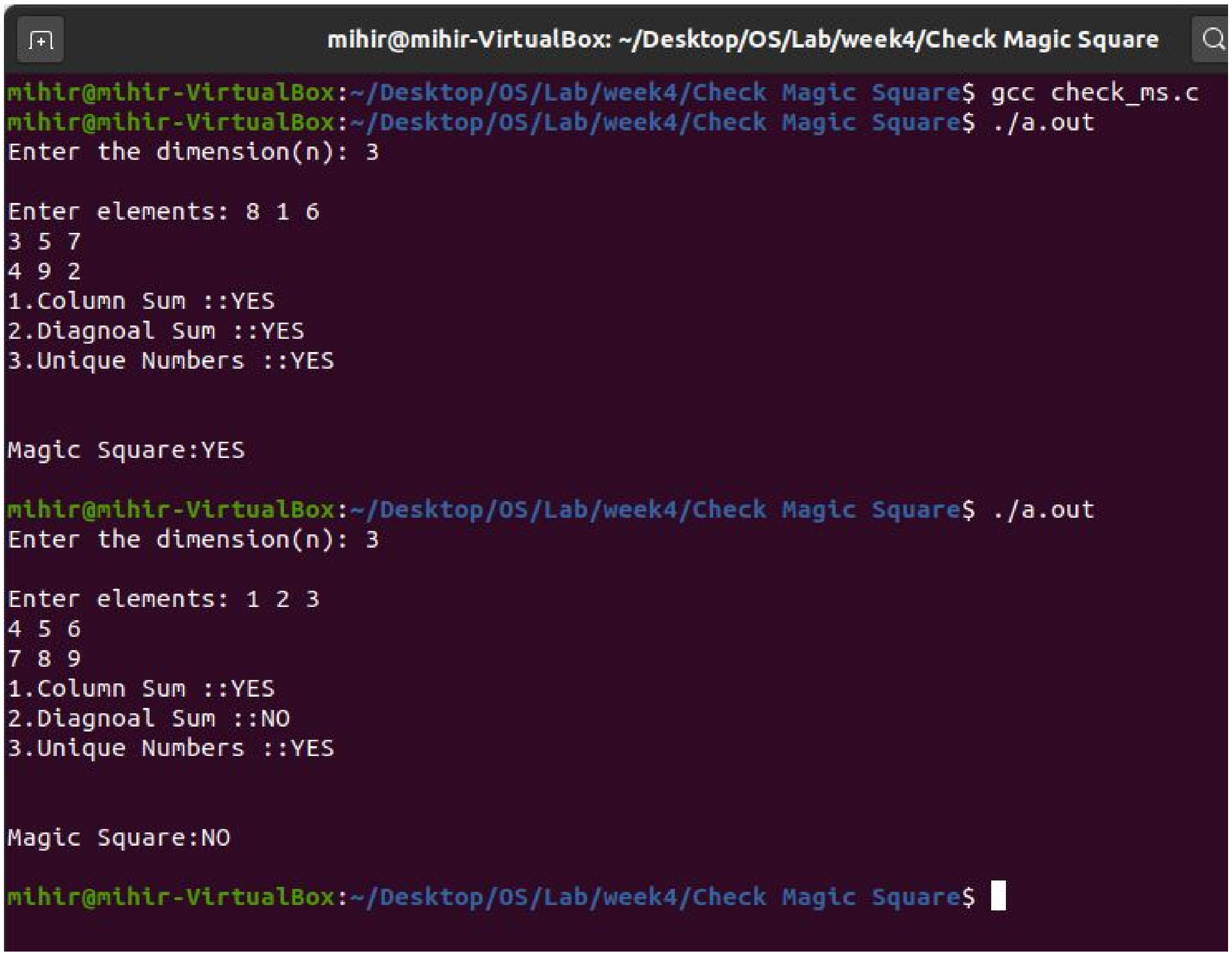
```
#include<stdio.h>
#include<stdlib.h>
#include<sys/wait.h>
#include<unistd.h>
#include<time.h>
int heap[10000];      //most efficient data structure to check uniqueness
int n;
int Colsum(int a[n][n]) {
        int count = 0;
        int countn = 0;
        for (int i = 0; i < n; i++)
                count += a[i][0];
        for (int j = 1; j < n; j++) {
                countn = 0;
                for (int i = 0; i < n; i++)
                countn += a[i][j];

                if (count != countn)
                return -1;
        }
        return count;
}

int Rowsum(int a[n][n]) {
        int count = 0;
        int countn = 0;
        for (int i = 0; i < n; i++)
                count += a[0][i];
        for (int j = 1; j < n; j++) {
                countn = 0;
                for (int i = 0; i < n; i++)
                countn += a[j][i];

                if (count != countn)
                return -1;
        }
        return count;
}
```

```
int Diagsum(int a[n][n]) {
        int countd1 = 0, countd2 = 0;
        for (int i = 0; i < n; i++) {
                countd1 += a[i][i];
                countd2 += a[i][n - i - 1];
        }

        if (countd1 == countd2)
                return countd1;
        else {
                return -1;
        }

}

int nocheck(int a[n][n]) {
        for (int i = 0; i < n; i++)
                heap[i] = 0;
        for (int i = 0; i < n; i++)
                for (int j = 0; j < n; j++) {
                if (heap[a[i][j]] != 1)
                        heap[a[i][j]] = 1;
                else
                        return -1;
                }
        return 1;
}

int main() {
        pid_t pid1, pid2, pid3;
        int sum = 0, status, flag = 0, x;
        printf("Enter the dimension(n): ");
        scanf("%d", &n);

        int a[n][n];

        printf("\nEnter elements: ");
```

```
for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
        scanf("%d", & a[i][j]);

pid1 = vfork();
if (pid1 == 0) {

        sum = Rowsum(a);
        _exit(0);

} else {

        pid2 = vfork();

        if (pid2 == 0) {
        printf("1.Column Sum :");
        if (Colsum(a) != sum) {
                printf(":NO\n");
                flag = 1;
        } else {
                printf(":YES\n");
        }

        _exit(0);
        }
        pid3 = vfork();
        if (pid3 == 0) {
        printf("2.Diagnoal Sum :");

        if (Diagsum(a) != sum) {
                printf(":NO\n");
                flag = 1;

        } else {
                printf(":YES\n");
        }
        _exit(0);
        } else {
        printf("3.Unique Numbers :");
```

```
                if (nocheck(a) == -1) {
                        printf(":NO\n");
                        flag = 1;
                } else {
                        printf(":YES\n");
                }
                }

        }
        waitpid(-1, & status, 0);
        printf("\n\nMagic Square");
        if (flag != 1)
                printf(":YES\n\n");
        else {
                printf(":NO\n\n");
        }

        return 0;
}
```

Output ss:


mihir@mihir-VirtualBox: ~/Desktop/OS/Lab/week4/Check Magic Square

```
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week4/Check Magic Square$ gcc check_ms.c
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week4/Check Magic Square$ ./a.out
Enter the dimension(n): 3

Enter elements: 8 1 6
3 5 7
4 9 2
1.Column Sum ::YES
2.Diagnoal Sum ::YES
3.Unique Numbers ::YES


Magic Square:YES

mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week4/Check Magic Square$ ./a.out
Enter the dimension(n): 3

Enter elements: 1 2 3
4 5 6
7 8 9
1.Column Sum ::YES
2.Diagnoal Sum ::NO
3.Unique Numbers ::YES


Magic Square:NO

mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week4/Check 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...)

Logic: We create different functions for different purposes.

Function1 (odd) - Generates magic square when the value of n is odd
Function2 (singly even) - Generates magic square when the value of n is of the form of 4*i + 2 where i >= 0.
Function3 (doubly even) - Generates magic square when the value of n is of the form of 4*i where i > 0.

Function4 (print) - A function that prints the magic square generated by one of the 3 functions above.
Function5 (check) - A function that checks whether the generated square is magic square or not.

We create two processes, one for generating the magic square and one for printing it.

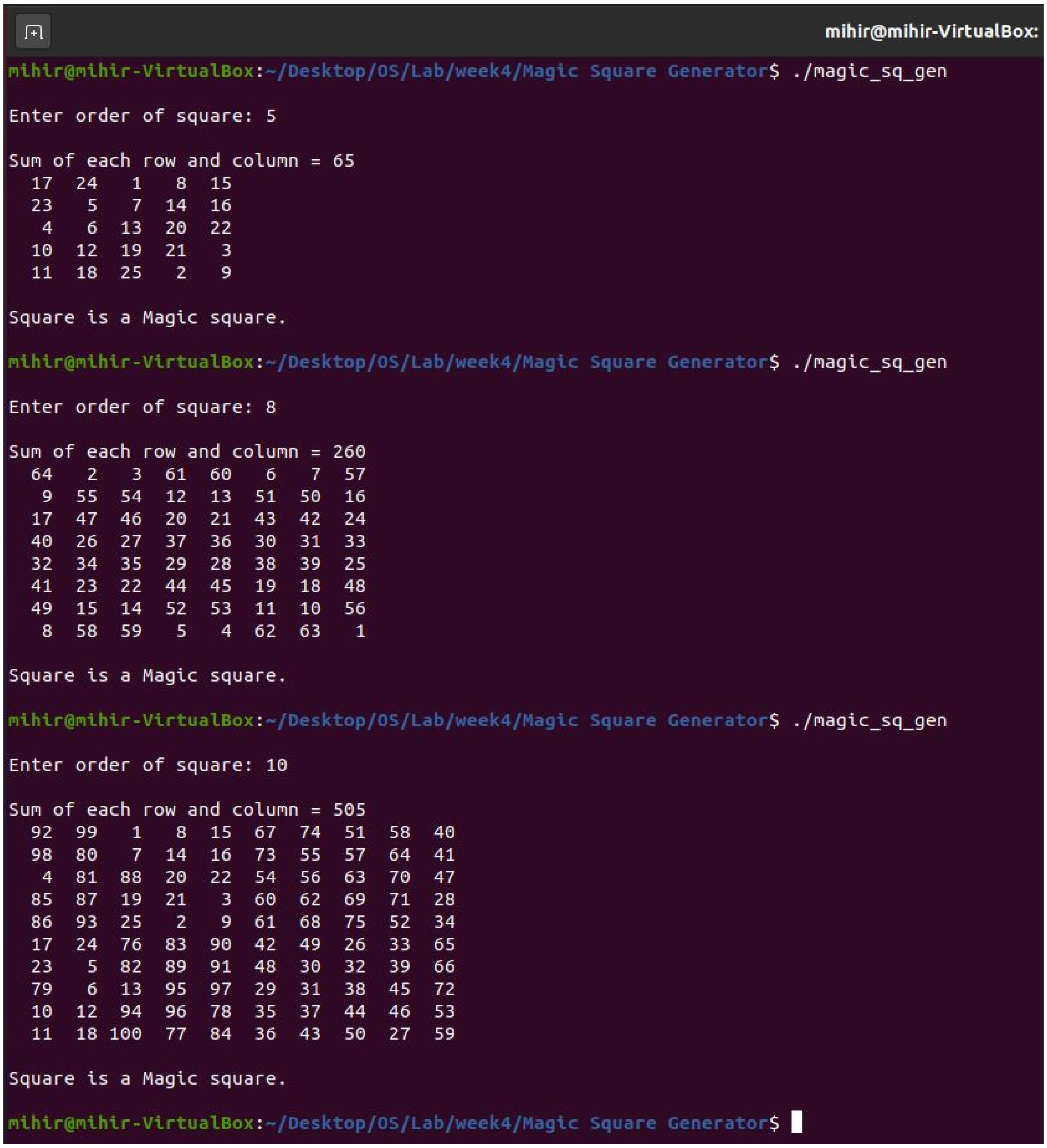
Code:
```
#include <iostream>
#include <unistd.h>
#include <vector>
#include <sys/wait.h>

using namespace std;

void OddMagicSquare(vector<vector<int> > &matrix, int n);
void DoublyEvenMagicSquare(vector<vector<int> > &matrix, int n);
void SinglyEvenMagicSquare(vector<vector<int> > &matrix, int n);
void MagicSquare(vector<vector<int> > &matrix, int n);
void PrintMagicSquare(vector<vector<int> > &matrix, int n);
int CheckSquare(vector<vector<int> > &matrix, int n);

int main(int argc, char* argv[])
{
  int n;
  printf("\nEnter order of square: ");
  scanf("%d", &n);

  vector<vector<int> > matrix(n, vector<int> (n, 0));

  pid_t pid = vfork();

  if (pid == 0)
  {
      if (n<3)
      {
                    printf("\nError: n must be greater than 2\n\n");
                    return -1;
```

```
        }

        MagicSquare(matrix, n);
        exit(0);
    }

    else if (pid > 0)
    {
        wait(NULL);
        //Print results
        PrintMagicSquare(matrix, n);
        int Square_valid = CheckSquare(matrix, n);

        if (Square_valid) printf("Square is a Magic square.\n\n");
            else printf("Square is NOT a Magic square.\n\n");
    }

    return 0;
}

void MagicSquare(vector<vector<int> > &matrix,int n)
{
    if (n%2==1)          //n is Odd
            OddMagicSquare(matrix, n);
    else            //n is even
            if (n%4==0)   //doubly even order
            DoublyEvenMagicSquare(matrix, n);
            else    //singly even order
            SinglyEvenMagicSquare(matrix, n);
}

void OddMagicSquare(vector<vector<int> > &matrix, int n)
{
    int nsqr = n * n;
    int i=0, j=n/2;        // start position

    for (int k=1; k<=nsqr; ++k)
    {
            matrix[i][j] = k;
            i--;
            j++;

            if (k%n == 0)
            {
            i += 2;
            --j;
            }
            else
            {
            if (j==n)
            j -= n;
            else if (i<0)
            i += n;
            }
    }
}

void DoublyEvenMagicSquare(vector<vector<int> > &matrix, int n)
{
    vector<vector<int> > I(n, vector<int> (n, 0));
    vector<vector<int> > J(n, vector<int> (n, 0));

    int i, j;

    //prepare I, J
    int index=1;
    for (i=0; i<n; i++)
            for (j=0; j<n; j++)
            {
            I[i][j]=((i+1)%4)/2;
            J[j][i]=((i+1)%4)/2;
            matrix[i][j]=index;
            index++;
            }

    for (i=0; i<n; i++)
            for (j=0; j<n; j++)
```

```
          {
          if (I[i][j]==J[i][j])
          matrix[i][j]=n*n+1-matrix[i][j];
          }
}

void SinglyEvenMagicSquare(vector<vector<int> > &matrix, int n)
{
  int p=n/2;

  vector<vector<int> > M(p, vector<int> (p, 0));
  MagicSquare(M, p);

  int i, j, k;

  for (i=0; i<p; i++)
          for (j=0; j<p; j++)
          {
          matrix[i][j]=M[i][j];
          matrix[i+p][j]=M[i][j]+3*p*p;
          matrix[i][j+p]=M[i][j]+2*p*p;
          matrix[i+p][j+p]=M[i][j]+p*p;
          }

  if (n==2)
          return;

  vector<int> I(p, 0);
  vector<int> J;

  for (i=0; i<p; i++)
          I[i]=i+1;

  k=(n-2)/4;

  for (i=1; i<=k; i++)
          J.push_back(i);

  for (i=n-k+2; i<=n; i++)
```

```
            J.push_back(i);

   int temp;
   for (i=1; i<=p; i++)
            for (j=1; j<=J.size(); j++)
            {
            temp=matrix[i-1][J[j-1]-1];
            matrix[i-1][J[j-1]-1]=matrix[i+p-1][J[j-1]-1];
            matrix[i+p-1][J[j-1]-1]=temp;
            }

   //j=1, i
   //i=k+1, k+1+p
   i=k;
   j=0;
   temp=matrix[i][j]; matrix[i][j]=matrix[i+p][j]; matrix[i+p][j]=temp;

   j=i;
   temp=matrix[i+p][j]; matrix[i+p][j]=matrix[i][j]; matrix[i][j]=temp;
}


void PrintMagicSquare(vector<vector<int> > &matrix, int n)
{
   printf("\nSum of each row and column = %d\n", n*(n*n + 1) / 2);

   for (int i=0; i<n; i++)
   {
            for (int j=0; j<n; j++)
            printf(" %3d", matrix[i][j]);

            printf("\n");
   }

     printf("\n");
}

int CheckSquare(vector<vector<int> > &matrix, int n)
{
     int suml = 0, sumr = 0;
     int sum = n*(n*n+1)/2;

     for (int i = 0; i < n; i++)
     {
             suml += matrix[i][i];
             sumr += matrix[i][n-1-i];
     }

     if (suml != sum || sumr != sum)
             return 0;

            for(int i=0;i<n;i++)
            {
            int rsum = 0, csum = 0;
            for(int j=0;j<n;j++)
            {
            rsum = rsum + matrix[i][j];
            csum = csum + matrix[j][i];
            }

            if (rsum != sum || csum != sum)
            return 0;
            }

            return 1;
}
```

Output ss:



```
mihir@mihir-VirtualBox:
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week4/Magic Square Generator$ ./magic_sq_gen

Enter order of square: 5

Sum of each row and column = 65
  17  24   1   8  15
  23   5   7  14  16
   4   6  13  20  22
  10  12  19  21   3
  11  18  25   2   9

Square is a Magic square.

mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week4/Magic Square Generator$ ./magic_sq_gen

Enter order of square: 8

Sum of each row and column = 260
  64   2   3  61  60   6   7  57
   9  55  54  12  13  51  50  16
  17  47  46  20  21  43  42  24
  40  26  27  37  36  30  31  33
  32  34  35  29  28  38  39  25
  41  23  22  44  45  19  18  48
  49  15  14  52  53  11  10  56
   8  58  59   5   4  62  63   1

Square is a Magic square.

mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week4/Magic Square Generator$ ./magic_sq_gen

Enter order of square: 10

Sum of each row and column = 505
  92  99   1   8  15  67  74  51  58  40
  98  80   7  14  16  73  55  57  64  41
   4  81  88  20  22  54  56  63  70  47
  85  87  19  21   3  60  62  69  71  28
  86  93  25   2   9  61  68  75  52  34
  17  24  76  83  90  42  49  26  33  65
  23   5  82  89  91  48  30  32  39  66
  79   6  13  95  97  29  31  38  45  72
  10  12  94  96  78  35  37  44  46  53
  11  18 100  77  84  36  43  50  27  59

Square is a Magic square.

mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week4/Magic Square Generator$ 
```