

OPERATING SYSTEMS - COM301P

LAB ASSIGNMENT - 4

FORKING ASSIGNMENT - 3

VIKNESH RAJARAMON

COE18B060

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

ORPHAN PROCESSES

LOGIC:

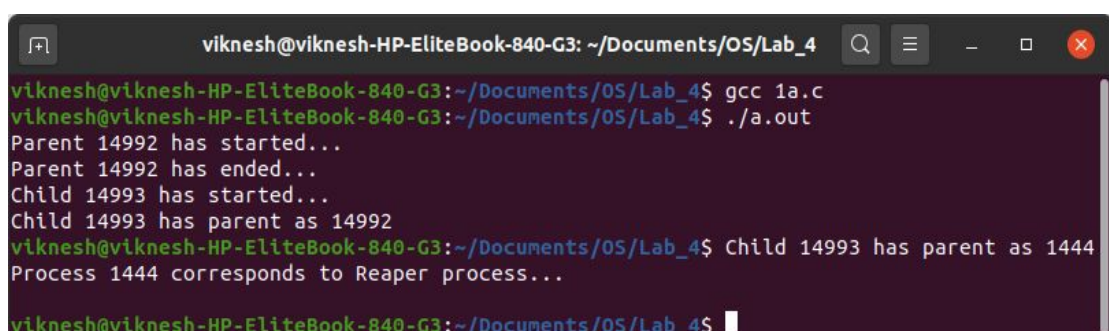
- Inside the Child process, use **sleep()** to suspend the execution of the child process. By this time, the parent process finishes its execution and therefore, the child process becomes an orphan process.
- When the parent process finishes its execution, the child is transferred to the **Reaper** process.

C CODE:

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

int main()
{
    pid_t pid;
    pid=fork();
    if(pid==0)
    {
        printf("Child %d has started...\n",getpid());
        printf("Child %d has parent as %d\n",getpid(),getppid());
        sleep(7);
        printf("Child %d has parent as %d\n",getpid(),getppid());
        printf("Process %d corresponds to Reaper process...\n",getppid());
    }
    else if(pid>0)
    {
        printf("Parent %d has started...\n",getpid());
        printf("Parent %d has ended...\n",getpid());
    }
    else
    {
        printf("Forking failed...\n");
    }
    return 0;
}
```

OUTPUT:



```
viknesh@viknesh-HP-EliteBook-840-G3: ~/Documents/OS/Lab_4
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_4$ gcc 1a.c
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_4$ ./a.out
Parent 14992 has started...
Parent 14992 has ended...
Child 14993 has started...
Child 14993 has parent as 14992
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_4$ Child 14993 has parent as 1444
Process 1444 corresponds to Reaper process...
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_4$
```

ZOMBIE PROCESSES

LOGIC:

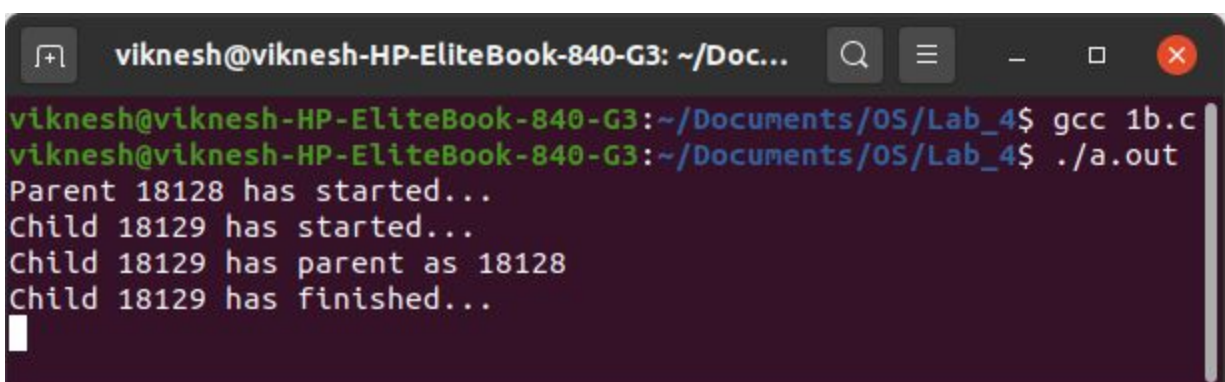
- Inside the Parent process, use **sleep()** to suspend the execution of the parent process. By this time, the child process finishes its execution and therefore, the child process becomes a zombie process.
- When the child process finishes its execution, it still has its entry in the **process table** so that it could report to its parent process.

C CODE:

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

int main()
{
    pid_t pid;
    pid=fork();
    if(pid==0)
    {
        printf("Child %d has started...\n",getpid());
        printf("Child %d has parent as %d\n",getpid(),getppid());
        printf("Child %d has finished...\n",getpid());
        exit(0);
    }
    else if(pid>0)
    {
        printf("Parent %d has started...\n",getpid());
        sleep(10);
        printf("Parent %d has finished...\n",getpid());
    }
    else
    {
        printf("Forking failed...\n");
    }
    return 0;
}
```

OUTPUT:



```
viknesh@viknesh-HP-EliteBook-840-G3: ~/Doc...
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_4$ gcc 1b.c
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_4$ ./a.out
Parent 18128 has started...
Child 18129 has started...
Child 18129 has parent as 18128
Child 18129 has finished...
```

```
viknesh@viknesh-HP-EliteBook-840-G3: ~/Doc...
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_4$ gcc 1b.c
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_4$ ./a.out
Parent 18128 has started...
Child 18129 has started...
Child 18129 has parent as 18128
Child 18129 has finished...
Parent 18128 has finished...
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_4$
```

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]

MERGE SORT

LOGIC:

- For performing **Merge sort()** parallely, we need to create a shared memory location for the array. We can do that by including the **<sys/shm.h>** header file.
- Once shared memory is created, we can implement left-side sort and right-side sort parallely. For creating parallel processes, we use **fork()**.
- We terminate the left-child process and right-child process once they are completed.
- In the parent process, we wait for both the children to finish and after that we merge both the left-side array and right-side array.
- The **Merge sort()** function is called recursively until the size of the array becomes '1' or '2'.
- If the size of the array is '1', we simply use **return**.
- If the size of the array is '2', we check if the element in the smaller index is greater than the element in the larger index. If it is **true**, then we swap both the elements and then use **return**. Otherwise, we simply use **return**.

C CODE:

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/wait.h>
#include<sys/shm.h>
#include<sys/ipc.h>
#include<unistd.h>
#include<time.h>

void merge(int a[],int low,int mid,int high)
{
    int size=high-low+1;
    int array[size];
    int i=low;
    int j=mid+1;
    int k=0;
    while(i<=mid && j<=high)
    {
        if(a[i]<a[j])
```

```

        {
            array[k]=a[i];
            ++i;
        }
        else
        {
            array[k]=a[j];
            ++j;
        }
        ++k;
    }
    while(i<=mid)
    {
        array[k]=a[i];
        ++i;
        ++k;
    }
    while(j<=high)
    {
        array[k]=a[j];
        ++j;
        ++k;
    }
    j=low;
    for(int i=0;i<size;++i)
    {
        a[j]=array[i];
        ++j;
    }
}

void Merge_Sort(int a[],int low,int high)
{
    int i;
    int length=high-low+1;
    if(length==1)
    {
        return;
    }
    else if(length==2)
    {
        if(a[low]>a[high])
        {
            int temp=a[low];
            a[low]=a[high];
            a[high]=temp;
        }
        return;
    }
    pid_t left_child_pid,right_child_pid;
    left_child_pid=fork();
    if(left_child_pid==0)
    {
        Merge_Sort(a,low,low+length/2-1);
        exit(0);
    }
    else if(left_child_pid>0)

```

```

{
    right_child_pid=fork();
    if(right_child_pid==0)
    {
        Merge_Sort(a,low+length/2,high);
        exit(0);
    }
    else if(right_child_pid<0)
    {
        printf("Right child process creation Failed...\n");
        exit(0);
    }
}
else
{
    printf("Left child process creation Failed...\n");
    exit(0);
}
int status;
waitpid(left_child_pid,&status,0);
waitpid(right_child_pid,&status,0);
merge(a,low,low+length/2-1,high);
}

```

```

int main()
{
    int n;
    printf("\nEnter the size of array : ");
    scanf("%d",&n);
    int shm_id;
    key_t key=IPC_PRIVATE;
    size_t SHM_SIZE=sizeof(int)*n;
    shm_id=shmget(key,SHM_SIZE,IPC_CREAT | 0666);
    if(shm_id<0)
    {
        printf("\nShared Memory Allocation Failed...\n");
        exit(0);
    }
    int *array;
    array=shmat(shm_id,NULL,0);
    if(array==(int *)-1)
    {
        printf("\nShared Memory Operations Failed...\n");
        exit(0);
    }
    srand(time(NULL));
    printf("\nArray elements before sorting : ");
    for(int i=0;i<n;++i)
    {
        array[i]=rand()%(3*n);
        printf("%d ",array[i]);
    }
    printf("\n");
    Merge_Sort(array,0,n-1);
    printf("\nArray elements after sorting : ");
    for(int i=0;i<n;++i)
    {

```

```

        printf("%d ",array[i]);
    }
    printf("\n");
    if(shmdt(array)==-1)
    {
        printf("\nDetaching Shared Memory Failed...\n");
        exit(0);
    }
    if(shmctl(shm_id,IPC_RMID,NULL)==-1)
    {
        printf("\nRemoving Shared Memory Failed...\n");
        exit(0);
    }
    return 0;
}

```

OUTPUT:

```

viknesh@viknesh-HP-EliteBook-840-G3: ~/Documents/OS/Lab_4
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_4$ gcc 2.c
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_4$ ./a.out

Enter the size of array : 20
Array elements before sorting : 56 7 17 55 22 9 37 30 52 29 29 24 14 33 48 26 5 45 43 53
Array elements after sorting : 5 7 9 14 17 22 24 26 29 29 30 33 37 43 45 48 52 53 55 56
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_4$ ./a.out

Enter the size of array : 25
Array elements before sorting : 40 25 64 45 52 74 44 40 2 52 58 64 73 50 73 5 4 59 10 21 34 16 64 13 25
Array elements after sorting : 2 4 5 10 13 16 21 25 25 34 40 40 44 45 50 52 52 58 59 64 64 64 73 73 74
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_4$ ./a.out

Enter the size of array : 10
Array elements before sorting : 17 5 8 5 6 23 15 9 22 16
Array elements after sorting : 5 5 6 8 9 15 16 17 22 23
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_4$

```

QUICK SORT

LOGIC:

- For performing **Quick sort()** parallelly, we need to create a shared memory location for the array. We can do that by including the **<sys/shm.h>** header file.
- Once shared memory is created, we can implement left-side sort and right-side sort parallelly. For creating parallel processes, we use **fork()**.
- We terminate the left-child process and right-child process once they are completed.
- In the parent process, we wait for both the children to finish.
- In **Quick sort()**, we need to **partition** the array with respect to the **pivot element**. In this case, we choose the **last element** in the array as the **pivot element**.
- Since **partition()** cannot occur parallelly, we need to **fork()** after the partition function returns the current position of the pivot element.
- Once **partition()** returns the current position of the pivot element, we can run Quick sort() for left-side array and right-side array parallelly.
- If the size of the array is less than or equal to '1', we do not perform any operation.

C CODE:

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/wait.h>
#include<sys/shm.h>
#include<sys/ipc.h>
#include<unistd.h>
#include<time.h>

void swap(int *a,int *b)
{
    int temp=*a;
    *a=*b;
    *b=temp;
}

int partition(int arr[],int low,int high)
{
    int pivot_element=arr[high];
    int i=low-1;
    for(int j=low;j<high;++j)
    {
        if(arr[j]<pivot_element)
        {
            ++i;
            swap(&arr[i],&arr[j]);
        }
    }
    swap(&arr[i+1],&arr[high]);
    return (i+1);
}

void Quick_Sort(int arr[],int low,int high)
{
    if(low<high)
    {
        int mid=partition(arr,low,high);
        pid_t left_child_pid,right_child_pid;
        left_child_pid=fork();
        if(left_child_pid==0)
        {
            Quick_Sort(arr,low,mid-1);
            exit(0);
        }
        else if(left_child_pid>0)
        {
            right_child_pid=fork();
            if(right_child_pid==0)
            {
                Quick_Sort(arr,mid+1,high);
                exit(0);
            }
            else if(right_child_pid<0)
            {
                printf("Right child process creation Failed...\n");
                exit(0);
            }
        }
    }
}
```



```

    }
    else
    {
        printf("Left child process creation Failed...\n");
        exit(0);
    }
    int status;
    waitpid(left_child_pid,&status,0);
    waitpid(right_child_pid,&status,0);
}
}

int main()
{
    int n;
    printf("\nEnter the size of array : ");
    scanf("%d",&n);
    int shm_id;
    key_t key=IPC_PRIVATE;
    size_t SHM_SIZE=sizeof(int)*n;
    shm_id=shmget(key,SHM_SIZE,IPC_CREAT | 0666);
    if(shm_id<0)
    {
        printf("\nShared Memory Allocation Failed...\n");
        exit(0);
    }
    int *array;
    array=shmat(shm_id,NULL,0);
    if(array==(int *)-1)
    {
        printf("\nShared Memory Operations Failed...\n");
        exit(0);
    }
    srand(time(NULL));
    printf("\nArray elements before sorting : ");
    for(int i=0;i<n;++i)
    {
        array[i]=rand()%(3*n);
        printf("%d ",array[i]);
    }
    printf("\n");
    Quick_Sort(array,0,n-1);
    printf("\nArray elements after sorting : ");
    for(int i=0;i<n;++i)
    {
        printf("%d ",array[i]);
    }
    printf("\n");
    if(shmdt(array)==-1)
    {
        printf("\nDetaching Shared Memory Failed...\n");
        exit(0);
    }
    if(shmctl(shm_id,IPC_RMID,NULL)==-1)
    {
        printf("\nRemoving Shared Memory Failed...\n");
        exit(0);
    }
}

```

```

    }
    return 0;
}

```

OUTPUT:

```

viknesh@viknesh-HP-EliteBook-840-G3: ~/Documents/OS/Lab_4
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_4$ gcc 2b.c
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_4$ ./a.out

Enter the size of array : 20
Array elements before sorting : 27 37 46 52 29 39 21 12 20 30 52 28 21 29 26 15 33 58 25 12
Array elements after sorting : 12 12 15 20 21 21 25 26 27 28 29 29 30 33 37 39 46 52 52 58
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_4$ ./a.out

Enter the size of array : 25
Array elements before sorting : 39 7 31 43 0 28 67 40 17 33 35 56 69 74 27 0 23 26 49 55 46 46 52 39 30
Array elements after sorting : 0 0 7 17 23 26 27 28 30 31 33 35 39 39 40 43 46 46 49 52 55 56 67 69 74
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_4$ ./a.out

Enter the size of array : 10
Array elements before sorting : 19 18 17 4 26 21 24 2 19 1
Array elements after sorting : 1 2 4 17 18 19 19 21 24 26
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_4$

```

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

LOGIC:

- For counting the number of processes, we need to create a shared memory location. We can do that by including the **<sys/shm.h>** header file.
- Initially **count=1**, as the parent **./a.out** will be running.
- Now, we call **fork()**. Inside the child process, run a while loop to keep calling **fork()** until fork fails. Inside the child block, increment the value of count by 1.
- Even though the child process finishes its execution, the value returned by the child process will be stored in the process table.
- When **fork()** fails, the control is returned back to the first parent process. At this stage we can print the value of count.

C CODE:

```

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

int main()
{
    int shm_id;
    key_t key=IPC_PRIVATE;
    size_t SHM_SIZE=sizeof(int);
    shm_id=shmget(key,SHM_SIZE,IPC_CREAT | 0666);
    if(shm_id<0)
    {

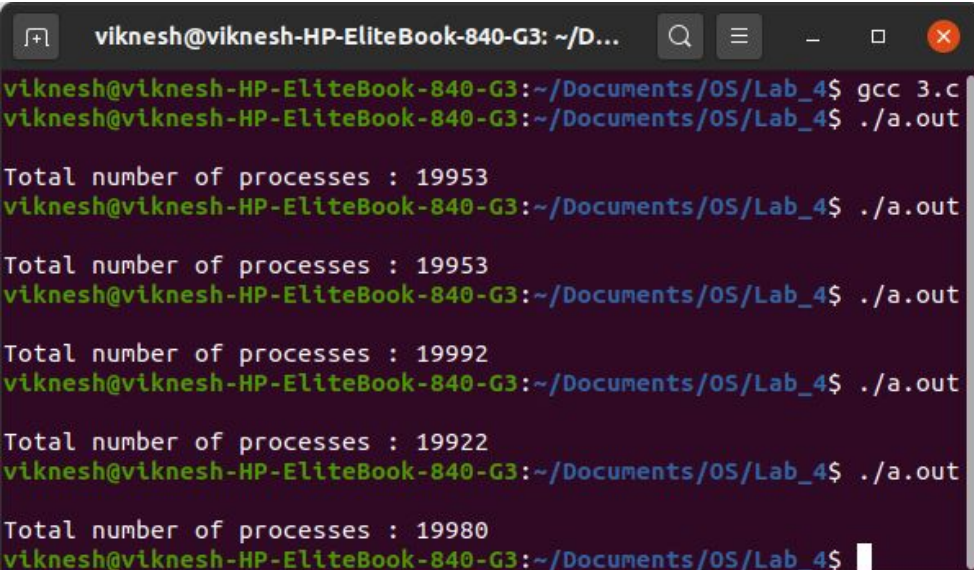
```

```

        printf("\nShared Memory Allocation Failed...\n");
        exit(0);
    }
    int *count;
    count=shmat(shm_id,NULL,0);
    if(count==(int *)-1)
    {
        printf("\nShared Memory Operations Failed...\n");
        exit(0);
    }
    (*count)=1;
    pid_t pid;
    int status;
    pid=fork();
    if(pid==0)
    {
        while(1)
        {
            pid=fork();
            if(pid==0)
            {
                (*count)=(*count)+1;
                exit(0);
            }
            else if(pid<0)
            {
                exit(0);
            }
        }
    }
    else if(pid>0)
    {
        wait(NULL);
    }
    else
    {
        printf("\nForking failed...\n");
        exit(0);
    }
    printf("\nTotal number of processes : %d\n",*count);
    if(shmdt(count)==-1)
    {
        printf("\nDetaching Shared Memory Failed...\n");
        exit(0);
    }
    if(shmctl(shm_id,IPC_RMID,NULL)==-1)
    {
        printf("\nRemoving Shared Memory Failed...\n");
        exit(0);
    }
    return 0;
}

```

OUTPUT:



```
viknesh@viknesh-HP-EliteBook-840-G3: ~/D...
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_4$ gcc 3.c
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_4$ ./a.out

Total number of processes : 19953
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_4$ ./a.out

Total number of processes : 19953
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_4$ ./a.out

Total number of processes : 19992
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_4$ ./a.out

Total number of processes : 19922
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_4$ ./a.out

Total number of processes : 19980
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_4$
```

- 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:

- We create a new terminal interface. This terminal doesn't support piping.
- **cd** command in Linux is implemented using **chdir()**.
- **close** command is used to exit from the terminal interface and return to the actual Linux terminal.
- **'&'** can be used to run in the background. This command uses **fork()** to create a parallel process. In the child process, the command is executed using **execvp()**.
- **history** command shows the recently entered **15 commands**.
- Most recent **N** commands can be shown using the **!N** command. **1 <= N <= 15** here. Otherwise, an error is thrown.
- If the command cannot be executed using **execvp()**, then an error is thrown.
- The interface has colours that are implemented using **\033**.

C CODE:

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

#define MAXSIZE 1024

char history[15][100];
int count=0;
```

```

void shell_prompt(char current_directory[])
{
    printf("\033[1;32m");
    printf("viknesh@OS-lab");
    printf("\033[0m");
    printf(":");
    printf("\033[1;34m");
    printf("%s",current_directory);
    printf("\033[0m");
    printf("$ ");
    fflush(stdout);
    return;
}

void display_history(int number)
{
    int x=count;
    int flag=1;
    if(number<x)
    {
        x=number;
        flag=0;
    }
    for(int i=0;i<x;++i)
    {
        printf(" %d. ",i+1);
        int j=0;
        while((history[i][j]!='\n') && (history[i][j]!='\0'))
        {
            printf("%c",history[i][j]);
            ++j;
        }
        printf("\n");
    }
    if(flag)
    {
        printf("No more commands in history\n");
    }
}

int command(char input_command[],char *arg[],int *flag)
{
    int length=read(STDIN_FILENO,input_command,MAXSIZE);
    for(int i=14;i>0;--i)
    {
        strcpy(history[i],history[i-1]);
    }
    strcpy(history[0],input_command);
    ++count;
    if(count>15)
    {
        count=15;
    }
    int start=-1;
    if(length==0)
    {

```

```

        exit(0);
    }
    else if(length<0)
    {
        printf("\nCommand not read\n");
        exit(0);
    }
    int j=0;
    for(int i=0;i<length;++i)
    {
        switch(input_command[i])
        {
            case ' ':
            case '\t':
                if(start!=-1)
                {
                    arg[j]=&input_command[start];
                    ++j;
                }
                input_command[i]='\0';
                start=-1;
                break;
            case '\n':
                if(start!=-1)
                {
                    arg[j]=&input_command[start];
                    ++j;
                }
                input_command[i]='\0';
                break;
            default:
                if(start==-1)
                {
                    start=i;
                }
                if(input_command[i]=='&')
                {
                    input_command[i]='\0';
                    *flag=1;
                }
                break;
        }
    }
    arg[j]=NULL;
    if(strcmp(arg[0],"close")==0)
    {
        exit(0);
    }
    if(strcmp(arg[0],"cd")==0)
    {
        if(chdir(arg[1])== -1)
        {
            printf("No such file or directory\n");
        }
        return -1;
    }
    if(strcmp(arg[0],"history")==0)

```

```

{
    if(count==0)
    {
        printf("No commands in history\n");
        return -1;
    }
    display_history(count);
    return -1;
}
else if(arg[0][0]=='!')
{
    if((arg[0][1]>='1') && (arg[0][1]<='9'))
    {
        if(arg[0][1]=='1')
        {
            if(arg[0][2]=='\0')
            {
                display_history(1);
            }
            else if((arg[0][2]>='0') && (arg[0][2]<='5') && (arg[0][3]=='\0'))
            {
                int x=arg[0][2]-'0';
                x=x+10;
                display_history(x);
            }
            else
            {
                printf("\nNo such command...Enter <= !15 (buffer size is 15 along with
current command)\n");
            }
        }
        else
        {
            if(arg[0][2]=='\0')
            {
                int x=arg[0][1]-'0';
                display_history(x);
            }
            else
            {
                printf("\nNo such command...Enter <= !15 (buffer size is 15 along with
current command)\n");
            }
        }
    }
    else
    {
        printf("\nNo such command...\n");
    }
    return -1;
}
}

int main()
{
    char current_directory[MAXSIZE];
    char input_command[100];

```

```

int flag=0,ret;
char *arg[MAXSIZE];
pid_t pid;
printf("\n");
while(1)
{
    flag=0;
    if(getcwd(current_directory,sizeof(current_directory))==NULL)
    {
        perror("getcwd() error\n");
        exit(0);
    }
    shell_prompt(current_directory);
    ret=command(input_command,arg,&flag);
    if(ret!=-1)
    {
        pid=fork();
        if(pid==0)
        {
            if(execvp(arg[0],arg)==-1)
            {
                printf("Error executing command\n");
            }
        }
        else if(pid<0)
        {
            printf("\nForking Failed...\n");
            exit(0);
        }
        else if(flag==0)
        {
            wait(NULL);
        }
    }
}
return 0;
}

```

OUTPUT:

The screenshot shows a terminal window with the following commands and output:

```

viknesh@viknesh-HP-EliteBook-840-G3: ~/Documents/OS/Lab_4$ gcc 4.c
viknesh@viknesh-HP-EliteBook-840-G3: ~/Documents/OS/Lab_4$ ./a.out

viknesh@OS-Lab: /home/viknesh/Documents/OS/Lab_4$ gedit sample.txt &
viknesh@OS-Lab: /home/viknesh/Documents/OS/Lab_4$ history
1. history
2. gedit sample.txt &
No more commands in history
viknesh@OS-Lab: /home/viknesh/Documents/OS/Lab_4$ close
viknesh@viknesh-HP-EliteBook-840-G3: ~/Documents/OS/Lab_4$

```

Below the terminal, a text editor window titled "sample.txt" is open, showing the text "1 Hello, How are you?". The status bar at the bottom indicates "Plain Text", "Tab Width: 8", "Ln 1, Col 20", and "INS" mode.


```
viknesh@viknesh-HP-EliteBook-840-G3: ~/Documents/OS/Lab_4
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_4$ gcc 4.c
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_4$ ./a.out

viknesh@OS-lab:/home/viknesh/Documents/OS/Lab_4$ ls
4.c  8.c  a.out  Done  forlabexercises03.pdf  sample.txt
viknesh@OS-lab:/home/viknesh/Documents/OS/Lab_4$ pwd
/home/viknesh/Documents/OS/Lab_4
viknesh@OS-lab:/home/viknesh/Documents/OS/Lab_4$ cat sample.txt
Hello, How are you?
viknesh@OS-lab:/home/viknesh/Documents/OS/Lab_4$ cd ..
viknesh@OS-lab:/home/viknesh/Documents/OS$ !15
1.  !15
2.  cd ..
3.  cat sample.txt
4.  pwd
5.  ls
No more commands in history
viknesh@OS-lab:/home/viknesh/Documents/OS$ !3
1.  !3
2.  !15
3.  cd ..
viknesh@OS-lab:/home/viknesh/Documents/OS$ !20
No such command...Enter <= !15 (buffer size is 15 along with current command)
viknesh@OS-lab:/home/viknesh/Documents/OS$ close
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_4$
```

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

LOGIC:

- For generating the **Histogram** parallelly, we need to create a common memory location for the frequency array. So, I have used the **<sys/shm.h>** header file to create a shared memory space for the array.
- Once shared memory is created, we can implement left-side sort and right-side sort parallelly. For creating parallel processes, we use **fork()**.
- We terminate the left-child process and right-child process once they are completed.
- In the parent process, we wait for both the children to finish and after that we merge both the left-side array and right-side array.
- The **Histogram generator()** function is called recursively until the size of the array becomes '1' or '2'.
- If the size of the array is '1', we count the frequency of the only character present in the array and then **return**.
- If the size of the array is '2', we count the frequency of the two characters present in the array and then **return**.

C CODE:

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

```

#include<sys/shm.h>
#include<sys/ipc.h>
#include<unistd.h>
#include<time.h>

#define MAXSIZE 1024

void Histogram_generator(char array[],char characters[],int freq[],int low,int high)
{
    int length=high-low+1;
    if(length==1)
    {
        ++freq[array[low]];
        return;
    }
    else if(length==2)
    {
        ++freq[array[low]];
        ++freq[array[high]];
        return;
    }
    pid_t left_array_pid,right_array_pid;
    left_array_pid=fork();
    if(left_array_pid==0)
    {
        Histogram_generator(array,characters,freq,low,low+length/2-1);
        exit(0);
    }
    else if(left_array_pid>0)
    {
        right_array_pid=fork();
        if(right_array_pid==0)
        {
            Histogram_generator(array,characters,freq,low+length/2,high);
            exit(0);
        }
        else if(right_array_pid<0)
        {
            printf("Right child process creation Failed...\n");
            exit(0);
        }
    }
    else
    {
        printf("Left child process creation Failed...\n");
        exit(0);
    }
    int status;
    waitpid(left_array_pid,&status,0);
    waitpid(right_array_pid,&status,0);
}

int main()
{
    char array[MAXSIZE];
    char characters[256];
    printf("\nEnter the text : ");

```

```

fgets(array,MAXSIZE,stdin);
int length=strlen(array);
int shm_id;
key_t key=IPC_PRIVATE;
size_t SHM_SIZE=sizeof(int)*(256);
shm_id=shmget(key,SHM_SIZE,IPC_CREAT | 0666);
if(shm_id<0)
{
    printf("\nShared Memory Allocation Failed...\n");
    exit(0);
}
int *frequency;
frequency=shmat(shm_id,NULL,0);
if(frequency==(int *)-1)
{
    printf("\nShared Memory Operations Failed...\n");
    exit(0);
}
for(int i=0;i<256;++i)
{
    characters[i]=i;
    frequency[i]=0;
}
Histogram_generator(array,characters,frequency,0,length-1);
printf("\n\t\tHistogram....\n");
printf("\nCharacter\tFrequency\tChart\n\n");
for(int i=0;i<256;++i)
{
    if((frequency[i]!=0) && (characters[i]!='\n'))
    {
        printf("[%c]\t\t%d\t\t",characters[i],frequency[i]);
        for(int j=0;j<frequency[i];++j)
        {
            printf("*");
        }
        printf("\n");
    }
}
if(shmdt(frequency)==-1)
{
    printf("\nDetaching Shared Memory Failed...\n");
    exit(0);
}
if(shmctl(shm_id,IPC_RMID,NULL)==-1)
{
    printf("\nRemoving Shared Memory Failed...\n");
    exit(0);
}
return 0;
}

```

OUTPUT:

```
viknesh@viknesh-HP-EliteBook-840-G3: ~/Documents/OS/Lab_4
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_4$ gcc 5.c
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_4$ ./a.out

Enter the text : hello, how are you? i am fine...

Histogram....

Character      Frequency      Chart
[ ]            6          *****
[, ]           1           *
[. ]           3          ***
[?]            1           *
[a]            2           **
[e]            3          ***
[f]            1           *
[h]            2           **
[i]            2           **
[l]            2           **
[m]            1           *
[n]            1           *
[o]            3          ***
[r]            1           *
[u]            1           *
[w]            1           *
[y]            1           *
```

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.

LOGIC:

- We fork **M*N** process, where M is the number of **rows in Matrix 1** and N is the number of **columns in Matrix 2**. We create a shared memory to store the resultant matrix.
- Each child process performs the multiplication of one row in **Matrix 1** with one column in **Matrix 2**.
- Finally, we wait for all the child processes to finish before displaying the output.

C CODE:

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

void Matrix_multiply(int m,int p,int n)
{
    srand(time(NULL));
```

```

int mat1[m][p],mat2[p][n];
printf("\nMatrix 1 : \n");
for(int i=0;i<m;++i)
{
    for(int j=0;j<p;++j)
    {
        mat1[i][j]=rand()%(m*n*3);
        printf("%d ",mat1[i][j]);
    }
    printf("\n");
}
printf("\nMatrix 2 : \n");
for(int i=0;i<p;++i)
{
    for(int j=0;j<n;++j)
    {
        mat2[i][j]=rand()%(m*n*3);
        printf("%d ",mat2[i][j]);
    }
    printf("\n");
}
int (*result)[m];
int shm_id;
key_t key=IPC_PRIVATE;
shm_id=shmget(key,sizeof(int[m][n]),IPC_CREAT | 0666);
if(shm_id<0)
{
    printf("\nShared Memory Allocation Failed...\n");
    exit(0);
}
result=shmat(shm_id,0,0);
if(result==(void *)-1)
{
    printf("\nShared Memory Operations Failed...\n");
    exit(0);
}
for(int i=0;i<m;++i)
{
    for(int j=0;j<n;++j)
    {
        pid_t pid=fork();
        if(pid==0)
        {
            result[i][j]=0;
            for(int k=0;k<p;++k)
            {
                result[i][j]+=mat1[i][k]*mat2[k][j];
            }
            exit(0);
        }
        else if(pid<0)
        {
            printf("\nForking Failed...\n");
            exit(0);
        }
    }
}
}

```

```

for(int i=0;i<m*n;++i)
{
    wait(NULL);
}
printf("Resultant Matrix : \n");
for(int i=0;i<m;++i)
{
    for(int j=0;j<n;++j)
    {
        printf("%d  ",result[i][j]);
    }
    printf("\n");
}
if(shmdt(result)==-1)
{
    printf("\nDetaching Shared Memory Failed...\n");
    exit(0);
}
if(shmctl(shm_id,IPC_RMID,NULL)==-1)
{
    printf("\nRemoving Shared Memory Failed...\n");
    exit(0);
}
return;
}

int main()
{
    int m1,n1,m2,n2;
    printf("\nEnter the dimensions of Matrix 1(separated by space) : ");
    scanf("%d %d",&m1,&n1);
    printf("\nEnter the dimensions of Matrix 2(separated by space) : ");
    scanf("%d %d",&m2,&n2);
    if(n1!=m2)
    {
        printf("\nMatrix 1 and Matrix 2 are not suitable for multiplication....\n");
        return 0;
    }
    Matrix_multiply(m1,n1,n2);
    return 0;
}

```

OUTPUT:

```
viknesh@viknesh-HP-EliteBook-840-G3: ~/D...
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_4$ gcc 6.c
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_4$ ./a.out

Enter the dimensions of Matrix 1(separated by space) : 3 3

Enter the dimensions of Matrix 2(separated by space) : 3 3

Matrix 1 :
9 3 22
4 18 19
15 24 12

Matrix 2 :
2 9 15
5 14 2
5 0 1

Resultant Matrix :
143 123 163
193 288 115
210 471 285
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_4$
```

```
viknesh@viknesh-HP-EliteBook-840-G3: ~/D...
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_4$ gcc 6.c
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_4$ ./a.out

Enter the dimensions of Matrix 1(separated by space) : 9 3

Enter the dimensions of Matrix 2(separated by space) : 3 6

Matrix 1 :
127 84 84
122 112 142
37 150 46
19 154 116
60 158 106
115 34 114
71 155 117
11 3 159
36 35 45

Matrix 2 :
148 67 114 117 49 53
56 9 3 36 46 153
82 65 0 53 125 12

Resultant Matrix :
30388 14725 14730 22335 20587 20591
35972 18412 14244 25832 28880 25306
17648 6819 4668 12167 14463 25463
20948 10199 2628 13915 22515 25961
26420 12332 7314 18326 23458 28626
28272 15421 13212 20721 21449 12665
28782 13757 8559 20088 25234 28882
14834 11099 1263 9822 20552 2950
10978 5652 4209 7857 8999 7803
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_4$
```


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.

LOGIC:

- We fork 4 processes. The value returned by the 4 child processes is stored in a shared memory.
- Fork 1 calculates the sum of elements in prime diagonal. Returns the sum.
- Fork 2 calculates the sum of elements in secondary diagonal. Returns the sum.
- Fork 3 calculates the sum of elements in each row and if they are equal, it returns the sum else it returns -1.
- Fork 4 calculates the sum of elements in each column if they are equal, it returns the sum else it returns -1.
- When all the child processes finish, we check if the values returned by them are equal. If they are equal, then it is a **MAGIC SQUARE**.

C CODE:

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/wait.h>
#include<sys/shm.h>
#include<sys/ipc.h>
#include<unistd.h>
#include<time.h>

#define MAX 1000

void Check_Magic_Square(int n)
{
    int a[n][n];
    printf("\nEnter the elements : \n");
    for(int i=0;i<n;++i)
    {
        for(int j=0;j<n;++j)
        {
            scanf("%d",&a[i][j]);
        }
    }
    printf("\n");
    int shm_id;
    key_t key=IPC_PRIVATE;
    size_t SHM_SIZE=sizeof(int)*(4);
    shm_id=shmget(key,SHM_SIZE,IPC_CREAT | 0666);
    if(shm_id<0)
    {
        printf("\nShared Memory Allocation Failed...\n");
        exit(0);
    }
    int *flag;
    flag=shmat(shm_id,NULL,0);
```



```

if(flag==(int *)-1)
{
    printf("\nShared Memory Operations Failed...\n");
    exit(0);
}
pid_t prime_diagnol,secondary_diagnol,rows,columns;
prime_diagnol=fork();
if(prime_diagnol==0)
{
    int prime_diagnol_sum=0;
    for(int i=0;i<n;++i)
    {
        prime_diagnol_sum=prime_diagnol_sum+a[i][i];
    }
    flag[0]=prime_diagnol_sum;
    exit(0);
}
else if(prime_diagnol>0)
{
    secondary_diagnol=fork();
    if(secondary_diagnol==0)
    {
        int secondary_diagnol_sum=0;
        for(int i=0;i<n;++i)
        {
            secondary_diagnol_sum+=a[n-1-i][i];
        }
        flag[1]=secondary_diagnol_sum;
        exit(0);
    }
    else if(secondary_diagnol>0)
    {
        rows=fork();
        if(rows==0)
        {
            int sum_of_rows[n];
            for(int i=0;i<n;++i)
            {
                sum_of_rows[i]=0;
                for(int j=0;j<n;++j)
                {
                    sum_of_rows[i]+=a[i][j];
                }
            }
            flag[2]=sum_of_rows[0];
            for(int i=1;i<n;++i)
            {
                if(sum_of_rows[i-1]!=sum_of_rows[i])
                {
                    flag[2]=-1;
                    break;
                }
            }
            exit(0);
        }
        else if(rows>0)
        {

```

```

        columns=fork();
        if(columns==0)
        {
            int sum_of_columns[n];
            for(int i=0;i<n;++i)
            {
                sum_of_columns[i]=0;
                for(int j=0;j<n;++j)
                {
                    sum_of_columns[i]+=a[j][i];
                }
            }
            flag[3]=sum_of_columns[0];
            for(int i=1;i<n;++i)
            {
                if(sum_of_columns[i-1]!=sum_of_columns[i])
                {
                    flag[3]=-1;
                    break;
                }
            }
            exit(0);
        }
        else if(columns<0)
        {
            printf("Columns process creation Failed...\n");
            exit(0);
        }
    }
    else
    {
        printf("Rows process creation Failed...\n");
        exit(0);
    }
}
else
{
    printf("Secondary Diagnol process creation Failed...\n");
    exit(0);
}
}
else
{
    printf("Prime Diagnol process creation Failed...\n");
    exit(0);
}
int status;
waitpid(prime_diagnol,&status,0);
waitpid(secondary_diagnol,&status,0);
waitpid(rows,&status,0);
waitpid(columns,&status,0);
if((flag[0]==flag[1]) && (flag[1]==flag[2]) && (flag[2]==flag[3]))
{
    printf("Magic Square\n");
}
else
{

```

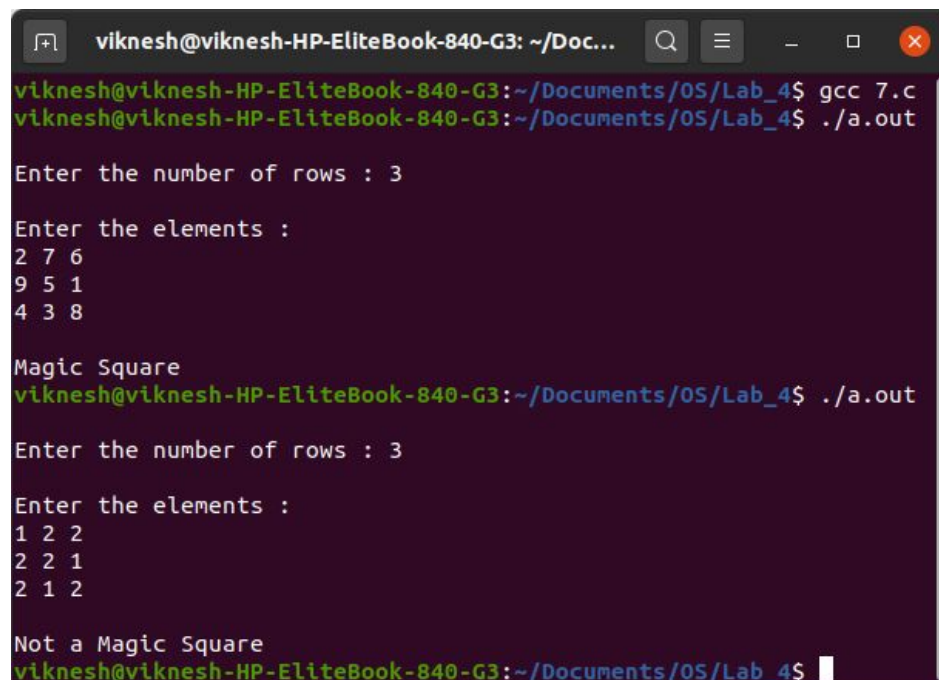
```

        printf("Not a Magic Square\n");
    }
    if(shmdt(flag)==-1)
    {
        printf("\nDetaching Shared Memory Failed...\n");
        exit(0);
    }
    if(shmctl(shm_id,IPC_RMID,NULL)==-1)
    {
        printf("\nRemoving Shared Memory Failed...\n");
        exit(0);
    }
}

int main()
{
    int n;
    printf("\nEnter the number of rows : ");
    scanf("%d",&n);
    Check_Magic_Square(n);
    return 0;
}

```

OUTPUT:



```

viknesh@viknesh-HP-EliteBook-840-G3: ~/Doc...
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_4$ gcc 7.c
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_4$ ./a.out

Enter the number of rows : 3

Enter the elements :
2 7 6
9 5 1
4 3 8

Magic Square
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_4$ ./a.out

Enter the number of rows : 3

Enter the elements :
1 2 2
2 2 1
2 1 2

Not a Magic Square
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_4$

```

```
viknesh@viknesh-HP-EliteBook-840-G3: ~/Doc...
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_4$ gcc 7.c
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_4$ ./a.out

Enter the number of rows : 10

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

Not a Magic Square
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_4$ ./a.out

Enter the number of rows : 10

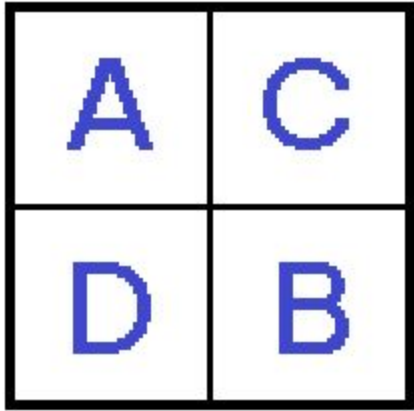
Enter the elements :
92 67 99 74 1 51 8 58 15 40
17 42 24 49 76 26 83 33 90 65
98 73 80 55 7 57 14 64 16 41
23 48 5 30 82 32 89 39 91 66
4 54 81 56 88 63 20 70 22 47
79 29 6 31 13 38 95 45 97 72
85 60 87 62 19 69 21 71 3 28
10 35 12 37 94 44 96 46 78 53
86 61 93 68 25 75 2 52 9 34
11 36 18 43 100 50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

Magic Square
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_4$
```

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:

- Magic squares are divided into 3 categories based on the order of the square. - Odd order ($n = 3, 5, 7, 9, \dots$), Doubly-even order ($n =$
- **Odd order** means that the order is an **odd number**. Examples are **N=3,5,7,9,...**
- **Singly-even order** means that the order is **divisible by 2** but **not divisible by 4**. Examples are **N=6,10,14,...**
- **Doubly-even order** means that the order is **divisible by 2 and 4**. Examples are **N=4,8,12,...**
- For **Odd order** magic square generation, we create two processes to fill the magic square. One process fills the square from **1 to $(N*N/2)+1$** while the other process fills the square from **$(N*N)$ to $(N*N/2)+2$** .
- For **Singly-even order** magic square generation, we first split the square into 4 equal halves like the one below:



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

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

Square A is a magic square with numbers from 1 to $(N*N/4)$. **Square B** is a magic square with numbers from $(N*N/4)+1$ to $(N*N/2)$. **Square C** is a magic square with numbers from $(N*N/2)+1$ to $(3*N*N/4)$. **Square D** is a magic square with numbers from $(3*N*N/4)+1$ to $(N*N)$.

- For **Doubly-even order** magic square generation, we first fill the array with their index count starting from 1. Change the value of elements at the top-left with the elements in the bottom-right. Also, change the value of elements at the top-right with the elements in the bottom-left. Do the same process in the center of the matrix.

C CODE:

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

```
#define MAXSIZE 100
```

```

void odd_order(int n,int a[][MAXSIZE])
{
    int i=n/2;
    int j=n-1;
    int p=n/2;
    int q=0;
    int split=((n*n)/2);
    pid_t pid1,pid2;
    pid1=fork();
    if(pid1==0)
    {
        for(int num=1;num<=split+1;)
        {
            if((i==-1) && (j==n))
            {
                i=0;
                j=n-2;
            }
            else
            {
                if(j==n)
                {
                    j=0;
                }
                if(i<0)
                {
                    i=n-1;
                }
            }
            if(a[i][j])
            {
                j-=2;
                ++i;
                continue;
            }
            else
            {
                a[i][j]=num;
                ++num;
            }
            ++j;
            --i;
        }
        exit(0);
    }
    else if(pid1>0)
    {
        pid2=fork();
        if(pid2==0)
        {
            for(int num=n*n;num>split+1;)
            {
                if((p==n) && (q==-1))
                {
                    p=0;
                    q=n-2;
                }
            }
        }
    }
}

```

```

        else
        {
            if(q==n-1)
            {
                q=n-1;
            }
            if(p>n-1)
            {
                p=0;
            }
        }
        if(a[p][q])
        {
            q+=2;
            --p;
            continue;
        }
        else
        {
            a[p][q]=num;
            --num;
        }
        --q;
        ++p;
    }
    exit(0);
}
else if(pid2<0)
{
    printf("\nForking failed...\n");
    exit(0);
}
}
else
{
    printf("\nForking failed...\n");
    exit(0);
}
int status;
waitpid(pid1,&status,0);
waitpid(pid2,&status,0);
return;
}

void singly_even_order(int n,int a[][MAXSIZE])
{
    odd_order(n/2,a);
    pid_t pid1,pid2,pid3;
    pid1=fork();
    if(pid1==0)
    {
        for(int i=n/2;i<n;++i)
        {
            for(int j=n/2;j<n;++j)
            {
                a[i][j]=a[i-n/2][j-n/2]+(n*n/4);
            }
        }
    }
}

```

```

        }
        exit(0);
    }
    else if(pid1>0)
    {
        pid2=fork();
        if(pid2==0)
        {
            for(int i=0;i<n/2;++i)
            {
                for(int j=n/2;j<n;++j)
                {
                    a[i][j]=a[i][j-n/2]+(2*n*n/4);
                }
            }
            exit(0);
        }
        else if(pid2>0)
        {
            pid3=fork();
            if(pid3==0)
            {
                for(int i=n/2;i<n;++i)
                {
                    for(int j=0;j<n/2;++j)
                    {
                        a[i][j]=a[i-n/2][j]+(3*n*n/4);
                    }
                }
                exit(0);
            }
            else if(pid3<0)
            {
                printf("\nForking failed...\n");
                exit(0);
            }
        }
        else
        {
            printf("\nForking failed...\n");
            exit(0);
        }
    }
    else
    {
        printf("\nForking failed...\n");
        exit(0);
    }
}
int status;
waitpid(pid1,&status,0);
waitpid(pid2,&status,0);
waitpid(pid3,&status,0);
int count;
pid1=fork();
if(pid1==0)
{
    for(int i=0;i<n/2;++i)

```



```

        {
            int j=-1;
            if(i==n/4)
            {
                ++j;
            }
            count=n/4;
            for(;count>0;++j,--count)
            {
                int temp=a[i][j+1];
                a[i][j+1]=a[i+n/2][j+1];
                a[i+n/2][j+1]=temp;
            }
        }
        exit(0);
    }
    else if(pid1>0)
    {
        pid2=fork();
        if(pid2==0)
        {
            count=n/4-1;
            while(count>0)
            {
                for(int i=0;i<n/2;++i)
                {
                    int temp=a[i][n-count];
                    a[i][n-count]=a[i+n/2][n-count];
                    a[i+n/2][n-count]=temp;
                }
                --count;
            }
            exit(0);
        }
        else if(pid2<0)
        {
            printf("\nForking failed...\n");
            exit(0);
        }
    }
    else
    {
        printf("\nForking failed...\n");
        exit(0);
    }
    waitpid(pid1,&status,0);
    waitpid(pid2,&status,0);
    return;
}

void doubly_even_order(int n,int a[][MAXSIZE])
{
    for(int i=0;i<n;++i)
    {
        for(int j=0;j<n;++j)
        {
            a[i][j]=(n*i)+j+1;

```

```

    }
}
pid_t top_left, top_right, bottom_left, bottom_right, center;
top_left=fork();
if(top_left==0)
{
    for(int i=0; i<n/4; ++i)
    {
        for(int j=0; j<n/4; ++j)
        {
            a[i][j]=(n*n+1)-a[i][j];
        }
    }
    exit(0);
}
else if(top_left>0)
{
    top_right=fork();
    if(top_right==0)
    {
        for(int i=0; i<n/4; ++i)
        {
            for(int j=3*(n/4); j<n; ++j)
            {
                a[i][j]=(n*n+1)-a[i][j];
            }
        }
        exit(0);
    }
    else if(top_right>0)
    {
        bottom_left=fork();
        if(bottom_left==0)
        {
            for(int i=3*(n/4); i<n; ++i)
            {
                for(int j=0; j<n/4; ++j)
                {
                    a[i][j]=(n*n+1)-a[i][j];
                }
            }
            exit(0);
        }
        else if(bottom_left>0)
        {
            bottom_right=fork();
            if(bottom_right==0)
            {
                for(int i=3*(n/4); i<n; ++i)
                {
                    for(int j=3*(n/4); j<n; ++j)
                    {
                        a[i][j]=(n*n+1)-a[i][j];
                    }
                }
                exit(0);
            }
        }
    }
}

```

```

        else if(bottom_right>0)
        {
            center=fork();
            if(center==0)
            {
                for(int i=(n/4);i<3*(n/4);++i)
                {
                    for(int j=n/4;j<3*(n/4);++j)
                    {
                        a[i][j]=(n*n+1)-a[i][j];
                    }
                }
                exit(0);
            }
            else if(center<0)
            {
                printf("\nForking failed...\n");
                exit(0);
            }
        }
        else
        {
            printf("\nForking failed...\n");
            exit(0);
        }
    }
    else
    {
        printf("\nForking failed...\n");
        exit(0);
    }
}
else
{
    printf("\nForking failed...\n");
    exit(0);
}
}
int status;
waitpid(top_left,&status,0);
waitpid(top_right,&status,0);
waitpid(bottom_left,&status,0);
waitpid(bottom_right,&status,0);
waitpid(center,&status,0);
return;
}

void print(int n,int a[][MAXSIZE])
{
    for(int i=0;i<n;++i)
    {
        for(int j=0;j<n;++j)

```

```

        {
            printf("%d ",a[i][j]);
        }
        printf("\n");
    }
}

int main()
{
    int n;
    printf("\nEnter the order of Magic square : ");
    scanf("%d",&n);
    int shm_id;
    key_t key=IPC_PRIVATE;
    shm_id=shmget(key,sizeof(int[n][n]),IPC_CREAT | 0666);
    int (*a)[n];
    if(shm_id<0)
    {
        printf("\nShared Memory Allocation Failed...\n");
        exit(0);
    }
    a=shmat(shm_id,0,0);
    if(a==(void *)-1)
    {
        printf("\nShared Memory Operations Failed...\n");
        exit(0);
    }
    if((n<=0) && (n==2))
    {
        printf("\nMagic square not possible\n");
    }
    else if(n==1)
    {
        printf("\nMagic square of Order %d\n",n);
        printf("-----\n");
        printf("1\n");
    }
    else if(n%2==1)
    {
        printf("\nMagic square of Order %d\n",n);
        printf("-----\n");
        odd_order(n,a);
        print(n,a);
    }
    else if(n%4==0)
    {
        printf("\nMagic square of Order %d\n",n);
        printf("-----\n");
        doubly_even_order(n,a);
        print(n,a);
    }
    else
    {
        printf("\nMagic square of Order %d\n",n);
        printf("-----\n");
        singly_even_order(n,a);
        print(n,a);
    }
}

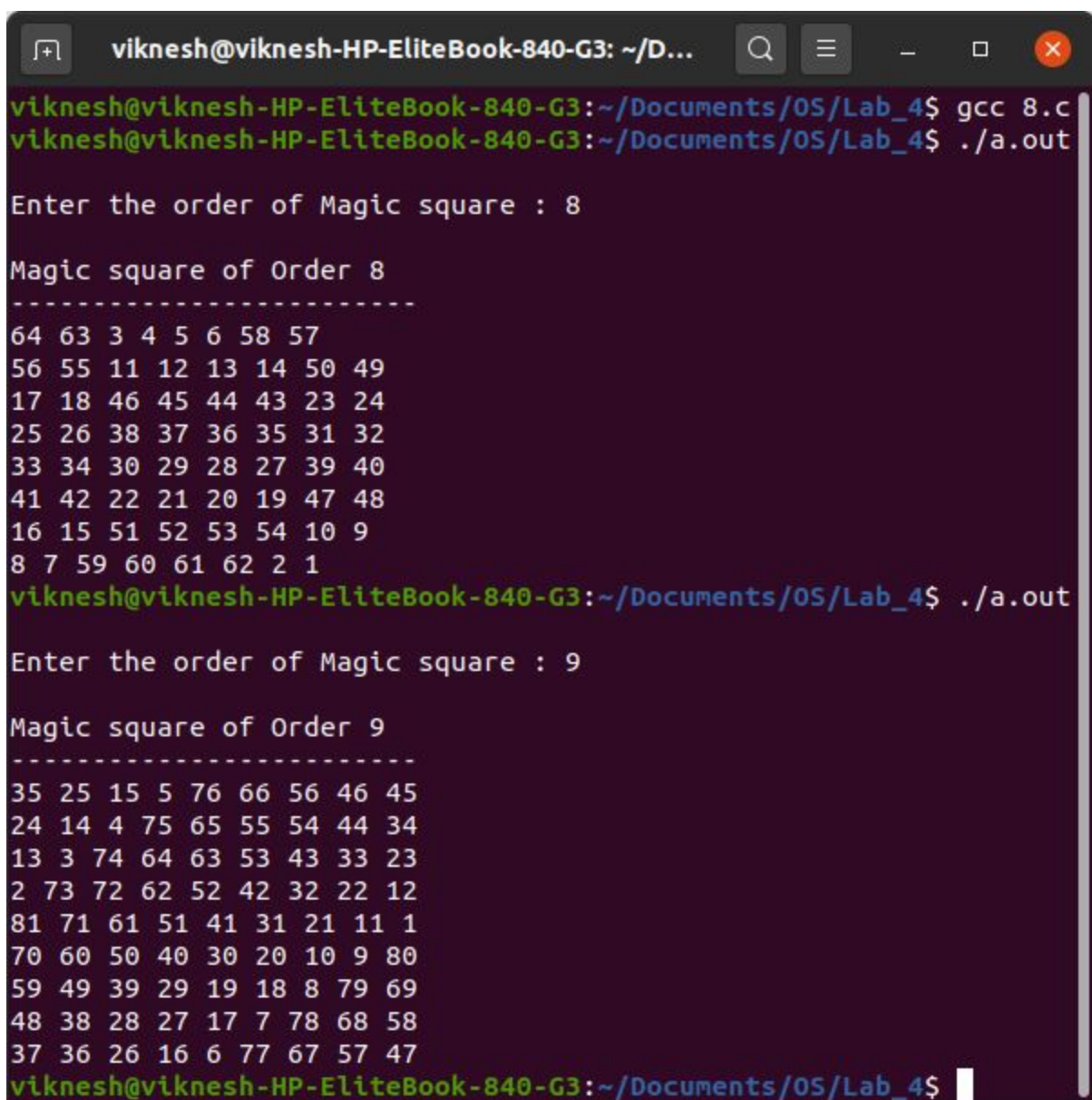
```

```

}
if(shmdt(a)==-1)
{
    printf("\nDetaching Shared Memory Failed...\n");
    exit(0);
}
if(shmctl(shm_id,IPC_RMID,NULL)==-1)
{
    printf("\nRemoving Shared Memory Failed...\n");
    exit(0);
}
return 0;
}

```

OUTPUT:



```

viknesh@viknesh-HP-EliteBook-840-G3: ~/D...
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_4$ gcc 8.c
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_4$ ./a.out

Enter the order of Magic square : 8

Magic square of Order 8
-----
64 63 3 4 5 6 58 57
56 55 11 12 13 14 50 49
17 18 46 45 44 43 23 24
25 26 38 37 36 35 31 32
33 34 30 29 28 27 39 40
41 42 22 21 20 19 47 48
16 15 51 52 53 54 10 9
8 7 59 60 61 62 2 1
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_4$ ./a.out

Enter the order of Magic square : 9

Magic square of Order 9
-----
35 25 15 5 76 66 56 46 45
24 14 4 75 65 55 54 44 34
13 3 74 64 63 53 43 33 23
2 73 72 62 52 42 32 22 12
81 71 61 51 41 31 21 11 1
70 60 50 40 30 20 10 9 80
59 49 39 29 19 18 8 79 69
48 38 28 27 17 7 78 68 58
37 36 26 16 6 77 67 57 47
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_4$

```

```
viknesh@viknesh-HP-EliteBook-840-G3: ~/D...  
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_4$ gcc 8.c  
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_4$ ./a.out  
  
Enter the order of Magic square : 10  
  
Magic square of Order 10  
-----  
84 78 22 16 15 59 53 72 66 40  
77 96 20 14 8 52 71 70 64 33  
25 94 88 7 1 75 69 63 57 26  
93 87 6 5 24 68 62 56 55 49  
86 85 4 23 17 61 60 54 73 42  
9 3 97 91 90 34 28 47 41 65  
2 21 95 89 83 27 46 45 39 58  
100 19 13 82 76 50 44 38 32 51  
18 12 81 80 99 43 37 31 30 74  
11 10 79 98 92 36 35 29 48 67  
viknesh@viknesh-HP-EliteBook-840-G3:~/Documents/OS/Lab_4$
```