FORK ASSIGNMENT - 3

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

Zombie Process:

A process which has finished the execution but still has entry in the process table to report to its parent process is known as a zombie process. A child process always first becomes a zombie before being removed from the process table. The parent process reads the exit status of the child process which reaps off the child process entry from the process table.

In the following code, the child finishes its execution using exit() system call while the parent sleeps for 50 seconds, hence doesn't call wait() and the child process's entry still exists in the process table.

Orphan Process:

A process whose parent process no longer exists i.e. either finished or terminated without waiting for its child process to terminate is called an orphan process.

```
//ZOMBIE
#include<stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
        pid_t child_pid = fork();

        if (child_pid > 0)
                sleep(50);

        else
                exit(0);

        return 0;
}
```

```
//ORPHAN
#include<stdio.h>
#include <sys/types.h>
#include <unistd.h>

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

        if (pid > 0)
                printf("in parent process");
```

```
	else if (pid == 0)
	{
		sleep(30);
		printf("in child process");
	}

	return 0;
}
```

---

(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]

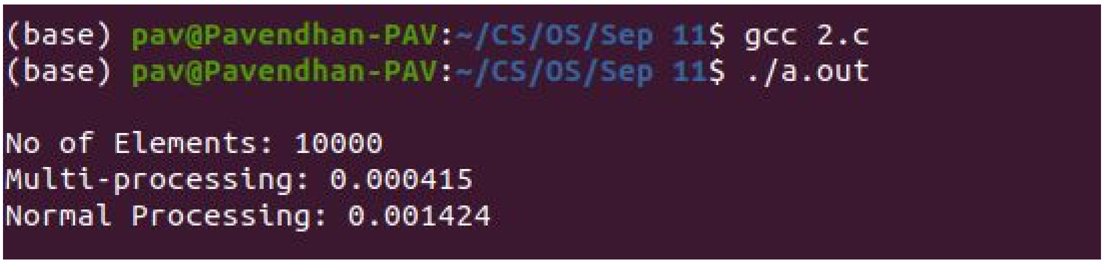
MERGE SORT

In the place where we normally execute the DIVIDE operation of the array into 2^n pieces for later CONQUER, we call the **vfork** for each divide operation, which in turn leads to parallelization of each conquer operation.

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


void merge(int start,int mid, int end,int arr[])
{
	int at[end+1],k=start;
	int i=start,j=mid+1;
	while(i<=mid && j<=end)
	{
		if(arr[i]<arr[j])
			at[k++]=arr[i++];
		else
		at[k++]=arr[j++];
	}

	if(i>mid)
		while(j<=end)
			at[k++]=arr[j++];

	if(j>end)
```

```
        while(i<=mid)
            at[k++]=arr[i++];

    for(int i=start;i<k;i++)
        arr[i]=at[i];
}

void msparallel(int start, int end,int arr[])
{
    if(start<end)
    {
        int mid=(start+end)/2;
        pid_t pid;
        pid=vfork();
        if(pid==0)
        {
            msparallel(start,mid,arr);
            _exit(0);
        }
        else
        {
            msparallel(mid+1,end,arr);
            merge(start,mid,end,arr);
        }
    }
}

void ms(int start, int end,int arr[])
{
    if(start<end)
    {
        int mid=(start+end)/2;
        ms(start,mid,arr);
        ms(mid+1,end,arr);
        merge(start,mid,end,arr);
    }
}

void main()
{
    int n;
    clock_t t1,t2;

    n=10000;
    int arr1[n];
    int arr2[n];
```

```
	printf("\nNo of Elements:%d\n",n);

	for(int i=0;i<10000;i++)
	{
		int x=rand();
		arr1[i]=arr2[i]=x;
	}

	t1=clock();
	msparallel(0,n-1,arr1);
	t2=clock();

	printf("Multi-processing%lf\n",(t2-t1)/(double)CLOCKS_PER_SEC);

	t1=clock();
	ms(0,n-1,arr2);
	t2=clock();

	printf("Normal Processing:%lf\n\n",(t2-t1)/(double)CLOCKS_PER_SEC);
}
```



```
(base) pav@Pavendhan-PAV:~/CS/OS/Sep 11$ gcc 2.c
(base) pav@Pavendhan-PAV:~/CS/OS/Sep 11$ ./a.out

No of Elements: 10000
Multi-processing: 0.000415
Normal Processing: 0.001424
```

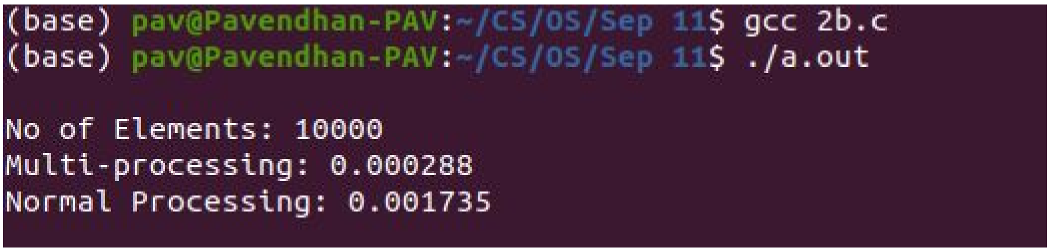
QUICK SORT

In the place where we normally execute the PARTITION and QUICKSORT for LEFT and RIGHT partitions, we call the **vfork** for each partition along with respective partition side sort, which in turn leads to parallelization of each partition side sort operation.

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


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

```
int partition (int arr[], int low, int high)
{
        int pivot = arr[high];
        int i = (low - 1);

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

/*
arr[] --> Array to be sorted,
low --> Starting index,
high --> Ending index */

void quickSortpar(int arr[], int low, int high)
{
        if (low < high)
        {
                /* pi is partitioning index, arr[p] is now
                at right place */


                pid_t pid;
                pid=vfork();
                if(pid==0)
                {
                        int pi = partition(arr, low, high);
                        quickSortpar(arr, low, pi - 1);
                        _exit(0);
                }
                else
                {
                        int pi = partition(arr, low, high);
                        quickSortpar(arr, pi + 1, high);
                }
        }
}
```

```
void quickSortser(int arr[], int low, int high)
{
    if (low < high)
    {
        /* pi is partitioning index, arr[p] is now
           at right place */
        int pi = partition(arr, low, high);

        quickSortser(arr, low, pi - 1);
        quickSortser(arr, pi + 1, high);
    }
}

void printArray(int arr[], int size)
{
            int i;
            for (i=0; i < size; i++)
                        printf("%d ", arr[i]);
            printf("\n");
}

int main()
{
            int n;
            clock_t t1,t2;

            n=10000;
            int arr1[n];
            int arr2[n];
            printf("\nNo of Elements: %d\n",n);

            for(int i=0;i<10000;i++)
            {
                        int x=rand();
                        arr1[i]=arr2[i]=x;
            }

            t1=clock();
            quickSortpar(arr1, 0, n-1);
            t2=clock();

            printf("Multi-processing: %lf\n",(t2-t1)/(double)CLOCKS_PER_SEC);

            t1=clock();
            quickSortser(arr2, 0, n-1);
            t2=clock();
            printf("Normal Processing: %lf\n\n",(t2-t1)/(double)CLOCKS_PER_SEC);

            return 0;
}
```



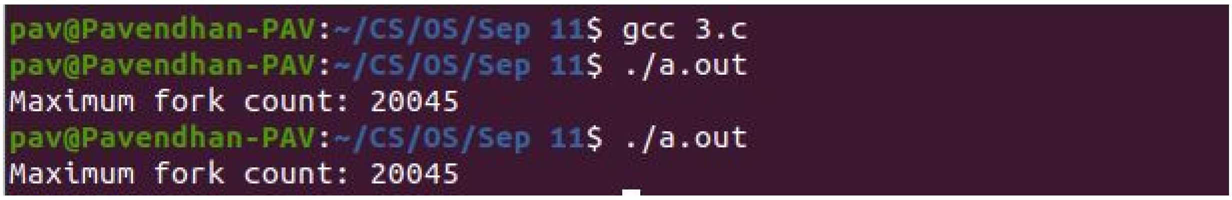
```
(base) pav@Pavendhan-PAV:~/CS/OS/Sep 11$ gcc 2b.c
(base) pav@Pavendhan-PAV:~/CS/OS/Sep 11$ ./a.out

No of Elements: 10000
Multi-processing: 0.000288
Normal Processing: 0.001735
```

---

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

Call fork repeatedly using for loop till fork starts to fail and exits for loop in the condition.

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

int main()
{
        long int count=0;
        int n=900000;

        for(int i=0; i<n; i++)
        {
                if(fork()==0)
                        exit(1);
        }

        for(int i=0; i<n; i++)
        {
                int pid;
                wait(&pid);
                pid /= 255; //the wait catches the child process's exit status 255 times
                count+=pid;
        }

        printf("Maximum fork count: %ld\n",count);
        return 0;
}
```



```
pav@Pavendhan-PAV:~/CS/OS/Sep 11$ gcc 3.c
pav@Pavendhan-PAV:~/CS/OS/Sep 11$ ./a.out
Maximum fork count: 20045
pav@Pavendhan-PAV:~/CS/OS/Sep 11$ ./a.out
Maximum fork count: 20045
```

---

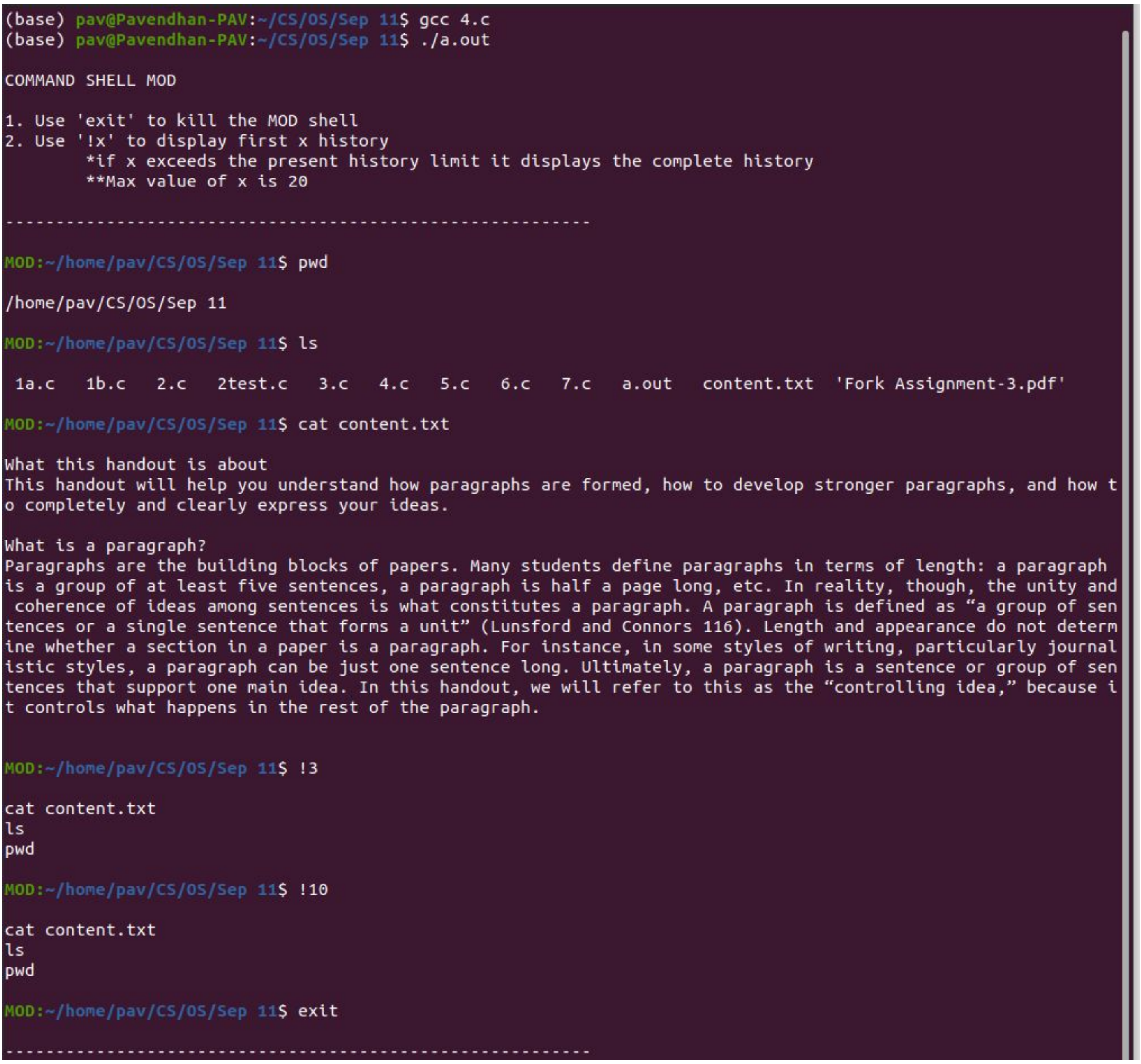
(4) Develop your own command shell [say mark it with @] that accepts user commands (System or User Binaries), executes the commands and returns the prompt for further user interaction. Also extend this to support a history feature (if the user types !6 at the command prompt; it should display the most 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.

Here the **execvp** command is used to execute the linux commands and history is created as an array where the given commands are stored and display those commands as soon as the command as explained in output is executed.

```
#include<stdio.h>
#include<unistd.h>
#include<string.h>
#include<stdlib.h>
#include<sys/wait.h>
#define maxcmd 20

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

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

void GetArgs(char c[])
{
        int i=0;

        arglen = 0;

        buflen = 0;
```

```
    int eol = 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]);
        }
}

int main()
{
        char cmd[500];
        char cwd[128];
        cmdno = 0;
        printf("\nCOMMAND SHELL MOD\n\n1. Use 'exit' to kill the MOD shell\n2. Use '!x' to display
first x history\n       *if x exceeds the present history limit it displays the complete history\n
**Max value of x is 20\n");
        printf("\n-----------------------------------------------------\n");

        while(1)
        {
                if(getcwd(cwd, sizeof(cwd))==NULL)
        {
                perror("getcwd() error");
                return 1;
        }

     printf("\n");
     printf("\033[1;32m");
     printf("MOD:");
     printf("\033[0m");
     printf("\033[1;34m");
     printf("~%s",cwd);
     printf("\033[0m");
     printf("$ ");
     fflush(stdout);
     scanf("%[^\n]%*c", cmd);


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

                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);
                            printf("\n");
                            if(execvp(argp[0], argp) == -1)
                            {
                                printf("%s: command not found.\n", argp[0]);
                            }
                            exit(0);
                        }

                        else
                        {
                            wait(NULL);
                        }
                }
        }
        s:  printf("\n---------------------------------------------------------\n");
        return 0;
}
```



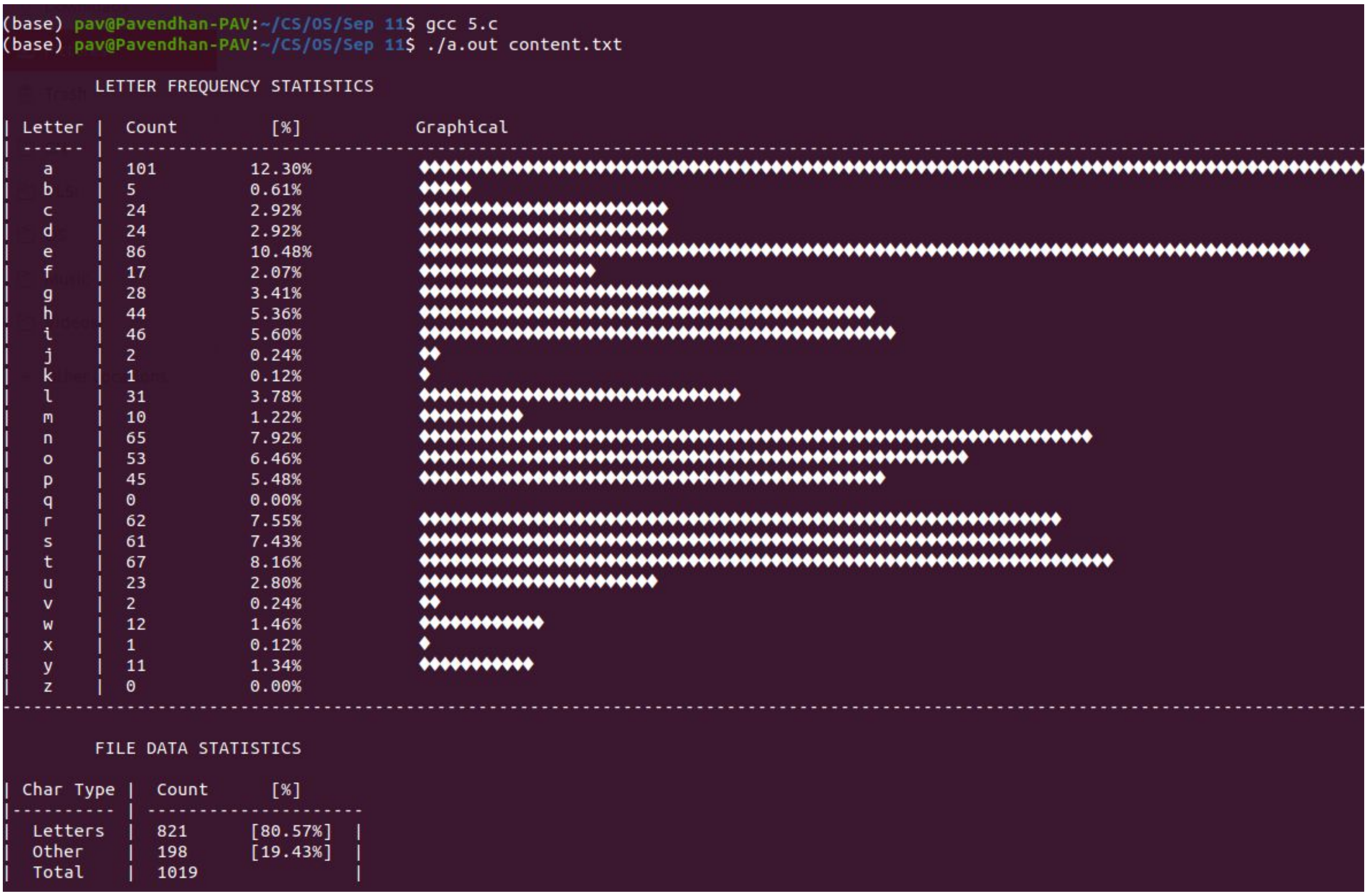
```
(base) pav@Pavendhan-PAV:~/CS/OS/Sep 11$ gcc 4.c
(base) pav@Pavendhan-PAV:~/CS/OS/Sep 11$ ./a.out

COMMAND SHELL MOD

1. Use 'exit' to kill the MOD shell
2. Use '!x' to display first x history
        *if x exceeds the present history limit it displays the complete history
        **Max value of x is 20

----------------------------------------------------------

MOD:~/home/pav/CS/OS/Sep 11$ pwd

/home/pav/CS/OS/Sep 11

MOD:~/home/pav/CS/OS/Sep 11$ ls

 1a.c   1b.c   2.c   2test.c   3.c   4.c   5.c   6.c   7.c   a.out   content.txt  'Fork Assignment-3.pdf'

MOD:~/home/pav/CS/OS/Sep 11$ cat content.txt

What this handout is about
This handout will help you understand how paragraphs are formed, how to develop stronger paragraphs, and how t
o completely and clearly express your ideas.

What is a paragraph?
Paragraphs are the building blocks of papers. Many students define paragraphs in terms of length: a paragraph
is a group of at least five sentences, a paragraph is half a page long, etc. In reality, though, the unity and
 coherence of ideas among sentences is what constitutes a paragraph. A paragraph is defined as “a group of sen
tences or a single sentence that forms a unit” (Lunsford and Connors 116). Length and appearance do not determ
ine whether a section in a paper is a paragraph. For instance, in some styles of writing, particularly journal
istic styles, a paragraph can be just one sentence long. Ultimately, a paragraph is a sentence or group of sen
tences that support one main idea. In this handout, we will refer to this as the “controlling idea,” because i
t controls what happens in the rest of the paragraph.


MOD:~/home/pav/CS/OS/Sep 11$ !3

cat content.txt
ls
pwd

MOD:~/home/pav/CS/OS/Sep 11$ !10

cat content.txt
ls
pwd

MOD:~/home/pav/CS/OS/Sep 11$ exit

----------------------------------------------------------
```

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

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

FILE *openFile(char *filename)
```

```
{
   FILE *file;
   file = fopen(filename, "r");

   if (!file)
   {
      printf("Error opening file!\n");
      return NULL;
   }

   return file;
}

void outputResults(int *charCount)
{
   long numLetters = 0;
   long totalChars = 0;

   for (int i = 32; i < 128; i++)
   {
      totalChars += charCount[i];
      if (i >= 97 && i <= 122)
         numLetters += charCount[i];
   }

   printf("\n\t LETTER FREQUENCY STATISTICS \n\n");
   printf("| Letter |  Count\t  [%%]\t\tGraphical\n");
   printf("| ------ |
----------------------------------------------------------------------------------------------------------------------------------------\n")
;

   for (int i = 97; i < 123; i++)
   {
      printf("|   %c    |  %0d ", i, charCount[i]);
      printf("  \t%.2f%%\t\t", ((double)charCount[i] / numLetters) * 100);
      for(int j=0; j<charCount[i]; j++)
         printf("♦");
      printf("\n");
   }


printf("------------------------------------------------------------------------------------------------------------------------------------
---------------\n");
   printf("\n\t FILE DATA STATISTICS \n\n");
   printf("| Char Type |  Count\t  [%%]\n");
   printf("|--------- | -------------------\n");
```

```
    printf("|  Letters  |  %li", numLetters);
    printf(" \t[%.2f%%]  |\n", ((double)numLetters / totalChars) * 100);
    printf("|  Other    |  %li", totalChars - numLetters);
    printf(" \t[%.2f%%]  |\n", ((double)(totalChars - numLetters) / totalChars) * 100);
    printf("|  Total    |  %li\t\t  |\n\n", totalChars);
}

int *countLetters(char *filename)
{
    int *charCount;
    FILE *file;

    charCount = mmap(NULL, 128 * sizeof(*charCount), PROT_WRITE, MAP_SHARED |
MAP_ANONYMOUS, -1, 0);

    for (int i = 0; i < 27; i++)
    {
        int c;

        if ((file = openFile(filename)) == NULL)
        {
            printf("Error opening file in child process %d!\n", getpid());
            exit(1);
        }

        pid_t pid = fork();

        if (pid == -1)
        {
            printf("Error forking process!\n");
            exit(1);
        }
        else if (pid == 0)
        {
            while ((c = tolower(fgetc(file))) != EOF)
            {
                if (i == 26 && (c < 97 || c > 122))
                    charCount[c]++; // Count other char
                else if (c == i + 97)
                    charCount[i + 97] += 1; // Count letters
            }

            fclose(file);
            exit(0);
        }
        else
            rewind(file);
    }

    for (int i = 0; i < 27; i++)
        wait(NULL);

    return charCount;
}

int main(int argc, char *argv[])
{
    if (argc != 2)
    {
        printf("Syntax: %s <filename>\n", argv[0]);
        return 1;
    }

    char *filename = argv[1];
    FILE *file;

    if ((file = openFile(filename)) == NULL)
        return 1;

    outputResults(countLetters(filename));

    if (fclose(file) != 0)
    {
        printf("Error closing file!\n");
        return 1;
    }

    return 0;
}
```



```
(base) pav@Pavendhan-PAV:~/CS/OS/Sep 11$ gcc 5.c
(base) pav@Pavendhan-PAV:~/CS/OS/Sep 11$ ./a.out content.txt

         LETTER FREQUENCY STATISTICS

| Letter |  Count        [%]           Graphical
| ------ | ------------------------------------------------------------------------------------------------
|   a    |  101        12.30%          ◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆
|   b    |  5          0.61%           ◆◆◆◆◆
|   c    |  24         2.92%           ◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆
|   d    |  24         2.92%           ◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆
|   e    |  86         10.48%          ◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆
|   f    |  17         2.07%           ◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆
|   g    |  28         3.41%           ◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆
|   h    |  44         5.36%           ◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆
|   i    |  46         5.60%           ◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆
|   j    |  2          0.24%           ◆◆
|   k    |  1          0.12%           ◆
|   l    |  31         3.78%           ◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆
|   m    |  10         1.22%           ◆◆◆◆◆◆◆◆◆◆
|   n    |  65         7.92%           ◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆
|   o    |  53         6.46%           ◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆
|   p    |  45         5.48%           ◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆
|   q    |  0          0.00%
|   r    |  62         7.55%           ◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆
|   s    |  61         7.43%           ◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆
|   t    |  67         8.16%           ◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆
|   u    |  23         2.80%           ◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆
|   v    |  2          0.24%           ◆◆
|   w    |  12         1.46%           ◆◆◆◆◆◆◆◆◆◆◆◆
|   x    |  1          0.12%           ◆
|   y    |  11         1.34%           ◆◆◆◆◆◆◆◆◆◆◆
|   z    |  0          0.00%
---------------------------------------------------------------------------------------------------------------

         FILE DATA STATISTICS

| Char Type |  Count      [%]
|---------- | --------------------
|  Letters  |  821       [80.57%]  |
|  Other    |  198       [19.43%]  |
|  Total    |  1019                |
```

(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)..

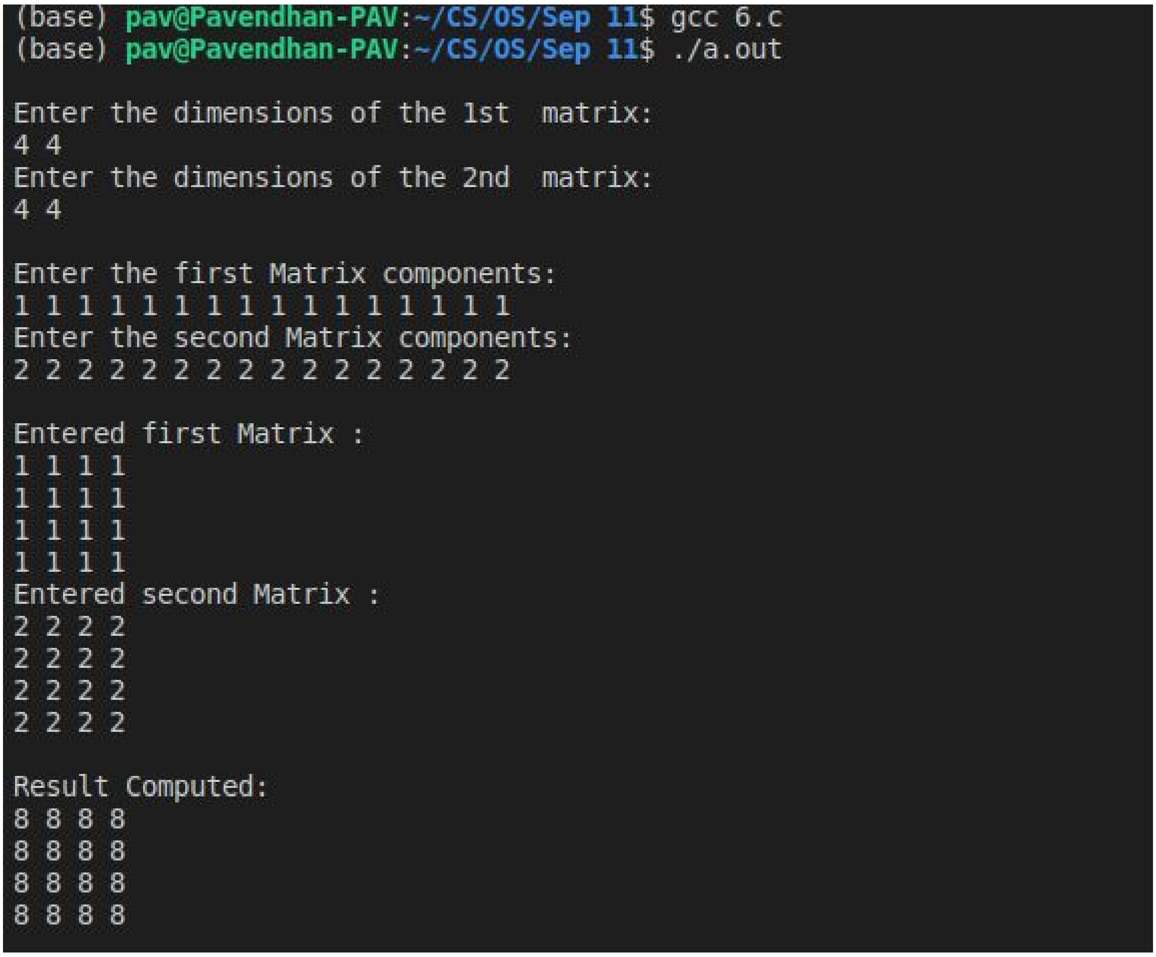
As stated in the question, each multiplication is parallelized in the most efficient way using **vfork()** where the data is shared across all the process and the overall output is accumulated and displayed in the end.

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

int r1, c1, r2, c2;

void get_input(int a, int b, int array[][b])
```

```
{
    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])
{
    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])
{
    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 dimensions of the 1st  matrix:\n");
    scanf("%d %d", &r1, &c1);
    printf("Enter the dimensions of the 2nd  matrix:\n");
    scanf("%d %d", &r2, &c2);

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

    if (c1 != r2)
    {
        printf("\nCannot Be Multiplied!!!\n");
        exit(0);
    }
```

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

printf("\nEntered first Matrix :\n");
display(r1, c1, a);
printf("Entered 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();
    if (pid[index++] == 0)
    {
      sum1 = matmul(i, j, a, b);
      c[i][j] = sum1;
      //gccprintf("%d ",sum1);

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

        //printf("%d ",sum2);
      }
    }
  }
}

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

```
    printf("\n");

    return 0;
}
```



```
(base) pav@Pavendhan-PAV:~/CS/OS/Sep 11$ gcc 6.c
(base) pav@Pavendhan-PAV:~/CS/OS/Sep 11$ ./a.out

Enter the dimensions of the 1st  matrix:
4 4
Enter the dimensions of the 2nd  matrix:
4 4

Enter the first Matrix components:
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
Enter the second Matrix components:
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2

Entered first Matrix :
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
Entered second Matrix :
2 2 2 2
2 2 2 2
2 2 2 2
2 2 2 2

Result Computed:
8 8 8 8
8 8 8 8
8 8 8 8
8 8 8 8
```

---

(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.

Here the **column, row & diagonal sum** sums are calculated using separate functions called through **vforks** and **_exit(0)** in if-else to share memory accordingly and finally check if the matrix is a magic matrix

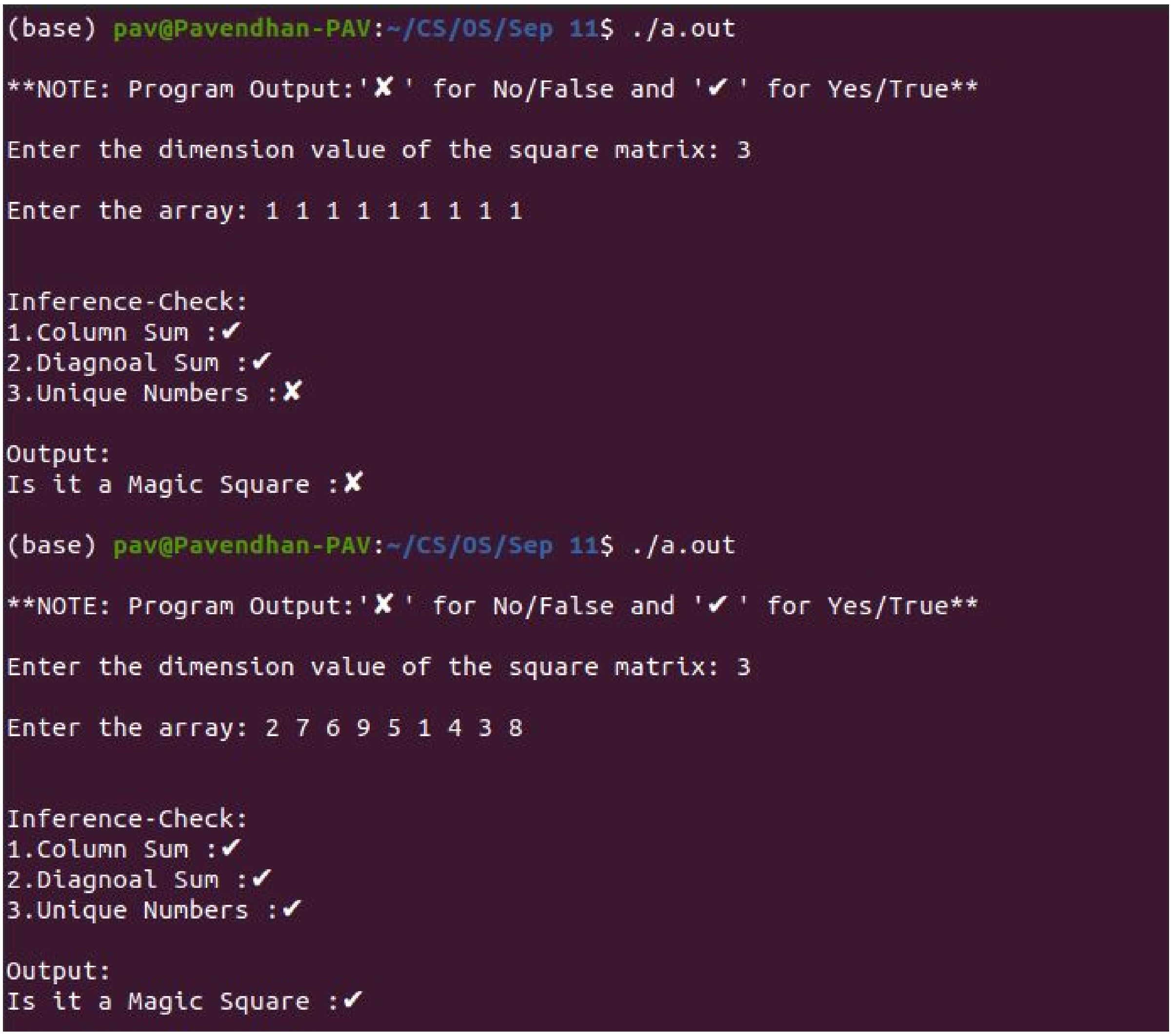
```
#include<stdio.h>
#include<stdlib.h>
#include<sys/wait.h>
#include<unistd.h>
#include<time.h>

int heap[10000];
const 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()
{
    printf("\n**NOTE: Program Output:\'✘ \' for No/False and \'✔ \' for Yes/True** \n\n");
    pid_t pid1,pid2,pid3;
    int sum=0,status,flag=0,x;
    printf("Enter the dimension value of the square matrix: ");
    scanf("%d",&n);

    int a[n][n];

    printf("\nEnter the array: ");

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

    printf("\n\nInference-Check:\n");
    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(" ✘ \n");
        flag=1;
      }
    else
    {
       printf("✔\n");
    }

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

    if(Diagsum(a)!=sum)
      {
        printf(" ✘ \n");
        flag=1;

      }
    else
    {
       printf("✔\n");
    }
    _exit(0);
  }
  else
  {
    printf("3.Unique Numbers :");
    if(nocheck(a)==-1)
     {
        printf(" ✘ \n");
```

```
            flag=1;
         }
        else
       {
          printf("✔\n");
       }
     }

   }
   waitpid(-1,&status,0);
   printf("\nOutput:\n");
      printf("Is it a Magic Square :");
   if(flag!=1)
      printf("✔\n\n");
   else
   {
      printf("✘\n\n");
   }

   return 0;
}
```



```
(base) pav@Pavendhan-PAV:~/CS/OS/Sep 11$ ./a.out

**NOTE: Program Output:'✘ ' for No/False and '✔ ' for Yes/True**

Enter the dimension value of the square matrix: 3

Enter the array: 1 1 1 1 1 1 1 1 1


Inference-Check:
1.Column Sum :✔
2.Diagnoal Sum :✔
3.Unique Numbers :✘

Output:
Is it a Magic Square :✘

(base) pav@Pavendhan-PAV:~/CS/OS/Sep 11$ ./a.out

**NOTE: Program Output:'✘ ' for No/False and '✔ ' for Yes/True**

Enter the dimension value of the square matrix: 3

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


Inference-Check:
1.Column Sum :✔
2.Diagnoal Sum :✔
3.Unique Numbers :✔

Output:
Is it a Magic Square :✔
```

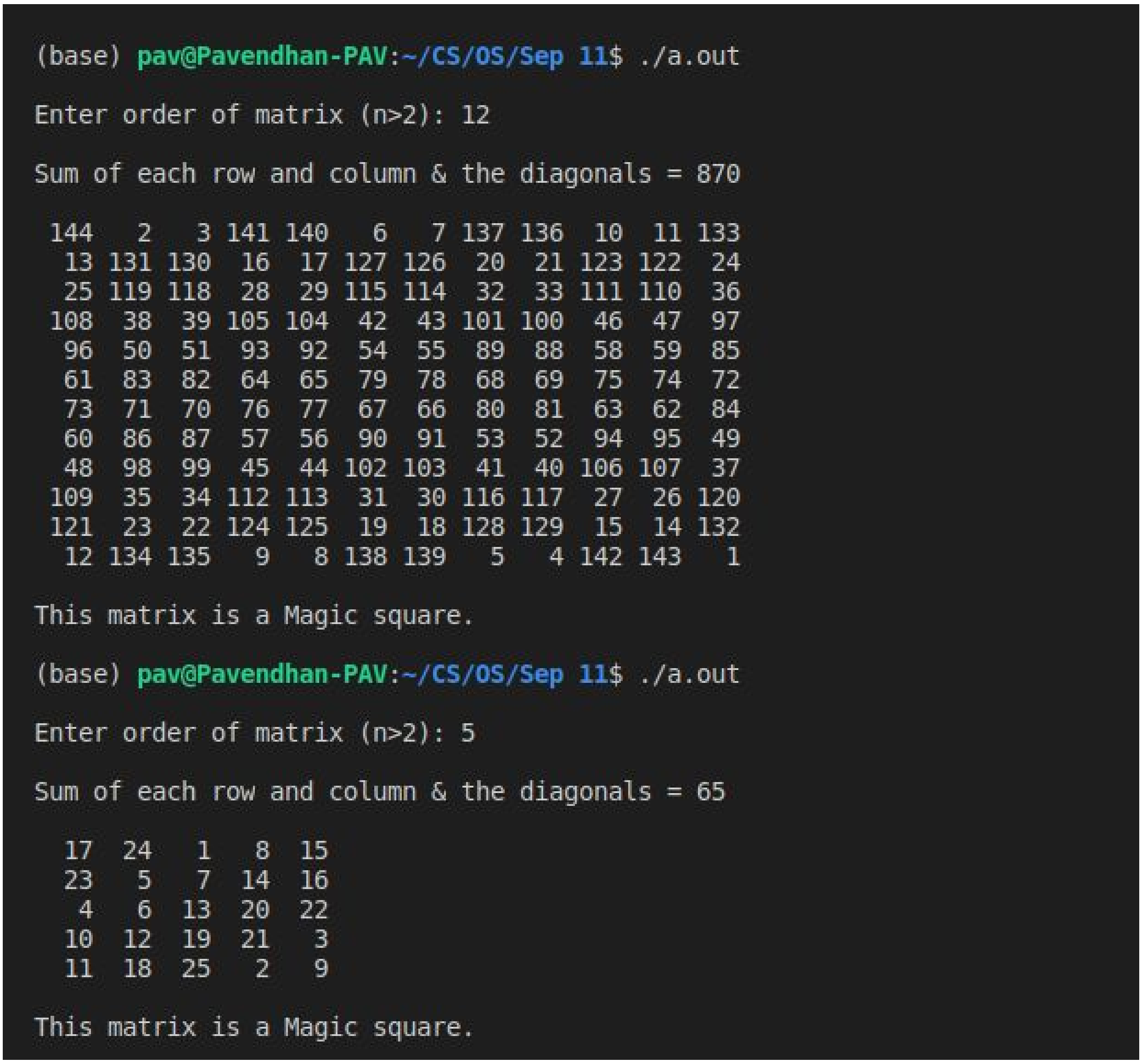
(8) Extend the above to also support magic square generation (u can take as input the order of the matrix..refer the net for algorithms for odd and even versions...)

Here there are important functions such as generation of magic squares and checking if this matrix is a magic square. These two here are parallelized using vfork where the buffer is shared between the processes leading to well balanced calculations in turn bringing up the magic square.

```
#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 matrix (n>2): ");
 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");
   exit(0);
  }

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

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

```
    PrintMagicSquare(matrix, n);
    int Square_valid = CheckSquare(matrix, n);

    if (Square_valid)
      printf("This matrix is a Magic square.\n\n");
    else
      printf("This matrix 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;

  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;

  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;
    }

  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 & the diagonals = %d\n\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;
}
```



```
(base) pav@Pavendhan-PAV:~/CS/OS/Sep 11$ ./a.out

Enter order of matrix (n>2): 12

Sum of each row and column & the diagonals = 870

 144   2   3 141 140   6   7 137 136  10  11 133
  13 131 130  16  17 127 126  20  21 123 122  24
  25 119 118  28  29 115 114  32  33 111 110  36
 108  38  39 105 104  42  43 101 100  46  47  97
  96  50  51  93  92  54  55  89  88  58  59  85
  61  83  82  64  65  79  78  68  69  75  74  72
  73  71  70  76  77  67  66  80  81  63  62  84
  60  86  87  57  56  90  91  53  52  94  95  49
  48  98  99  45  44 102 103  41  40 106 107  37
 109  35  34 112 113  31  30 116 117  27  26 120
 121  23  22 124 125  19  18 128 129  15  14 132
  12 134 135   9   8 138 139   5   4 142 143   1

This matrix is a Magic square.

(base) pav@Pavendhan-PAV:~/CS/OS/Sep 11$ ./a.out

Enter order of matrix (n>2): 5

Sum of each row and column & the diagonals = 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

This matrix is a Magic square.
```