COM301P Dr. Sivaselvan

# Assignment - 4
# Fork Assignment

By: Animesh Kumar (CED18I065)

Date: 04/10/2020

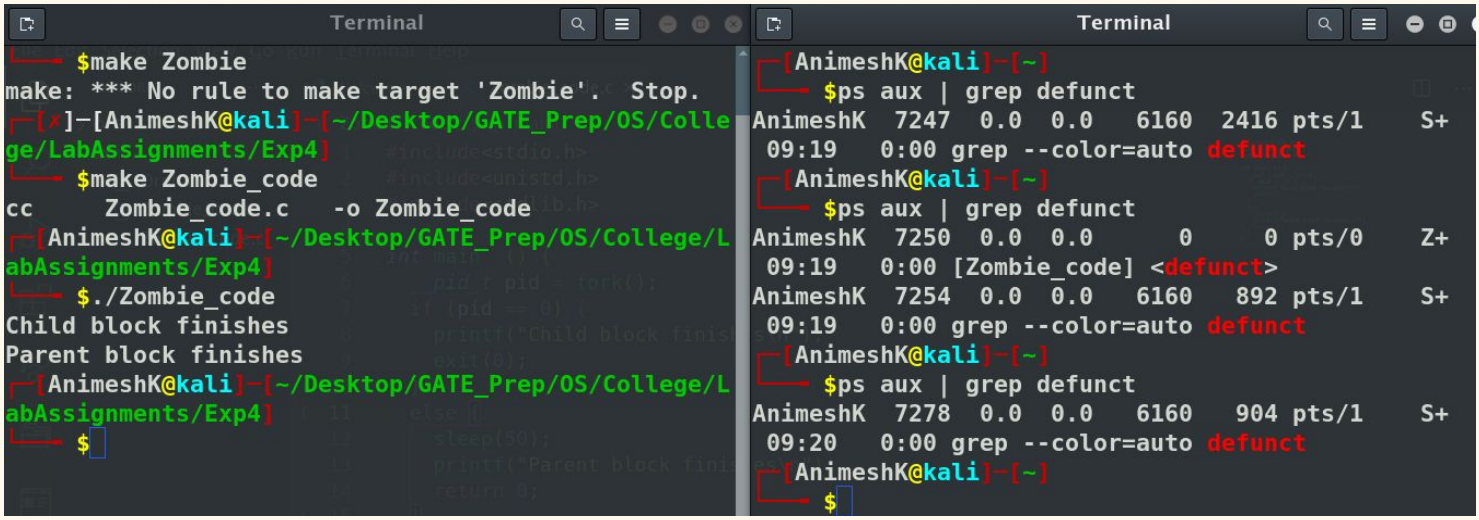
## 1. Zombie and Orphan:

- **Zombie:** The child process has terminated after executing some statements and since the parent process is sleeping for 50 seconds, it is still making its entry in its process table. In this case the child process becomes Zombie. When the parent gets to know the exit status of the child process then the parent removes that "Zombie entry" from its process table and this action is called 'reaping'.
  - C Program:

```c
#include<stdio.h>

#include<unistd.h>

#include<stdlib.h>

int main  () {

 __pid_t pid = fork();

 if (pid == 0) {

   printf("Child block finishes\n");

   exit(0);

 }

 else {
```

```
    sleep(50);

    printf("Parent block finishes\n");

    return 0;

 }

}
```

- Output:




- **Orphan:** The parent process terminates before the termination of child process, therefore the child process becomes orphaned. In my linux OS, the child process adopts the bash terminal as its new parent after the termination of the original parent.
  - C Program:

```
#include<stdio.h>

#include<unistd.h>

#include<stdlib.h>

int main  () {
```

```
 __pid_t pid = fork();

 if (pid == 0) {

   printf("I am Child(PID: %d) of my original parent: %d\n",
getpid(), getppid());

   sleep(20);

   printf("I am Child(PID: %d) of my current parent: %d\n", getpid(),
getppid());

   printf("Child block slept 20 secs -> finishes\n");

   exit(0);

 }

 else {

   sleep(5);

   printf("Parent block(PID: %d) slept 5 secs -> finishes and The
child PID: %d is still running\n", getpid(), pid);

   return 0;

 }

}
```

- Output:

```
┌──[✗]─[AnimeshK@kali]─[~/Desktop/GATE_Prep/OS/College/LabAssignments/Exp4]
└─$make Orphan_code
cc     Orphan_code.c   -o Orphan_code
┌──[AnimeshK@kali]─[~/Desktop/GATE_Prep/OS/College/LabAssignments/Exp4]
└─$./Orphan_code
I am Child(PID: 10007) of my original parent: 10006
Parent block(PID: 10006) slept 5 secs -> finishes and The child PID: 10007 is still running
┌──[AnimeshK@kali]─[~/Desktop/GATE_Prep/OS/College/LabAssignments/Exp4]
└─$I am Child(PID: 10007) of my current parent: 1313
Child block slept 20 secs -> finishes
```

## 2. Develop a multiprocessing version of Merge or Quick Sort.

- **Algorithm: The Merge() routine is inherently serial but MergeSort() is parallelized using child processes as in below pseudocode:**
  - Create Two pipes and fork() two child processes from the parent
  - The first child process
    - Recursively sorts the first half of the array.
    - Sends the half-sorted array to parent process via pipe
    - Terminate the child process successfully
  - The second child process
    - Recursively sorts the other half of the array.
    - Sends the half-sorted array to parent process via pipe
    - Terminate the child process successfully
  - The parent process
    - Waits for both the children to complete
    - Recieves both half sorted arrays from both the children via pipes.
    - Copies the relevant content to the final **int arr[]** array**.** Now the first and second halves of **arr[]** are sorted but the whole array is not.

- Calls Merge() routine to merge the sorted halves of the arr[]
- Terminate the parent process successfully

- **C Program:**

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

void MergeSort(int arr[], int l, int r, int size);
void merge(int arr[], int l, int m, int r);

int main () {
 int n;
 printf("Enter Array size: ");
 scanf("%d", &n);
 int arr[n];
 for (int i = 0; i < n; i++)
   scanf("%d", &arr[i]);
```

```
 MergeSort(arr, 0, n - 1, n);
  printf("\nSorted array is \n");
 for (int i = 0; i < n; i++)
   printf("%d ", arr[i]);
 printf("\n");
 return 0;
}


// The Multiprocessing version of MergeSort
void MergeSort(int arr[], int l, int r, int size)
{
 if (l < r) {
   int m = l + (r - l) / 2;
   __pid_t pid1, pid2, wpid;
   int fd1[2], fd2[2];  // file descriptors to hold return values of
pipe
   int status = 0;
   pipe(fd1);  // Pipe #1, for first child process IPC to parent
   pipe(fd2);  // Pipe #2, for second child process IPC to parent


   // forking two child processes
   ((pid1 = fork()) && (pid2 = fork()));
```

```
// Child Process #1
if (pid1 == 0) {
  // Sorting the first half of the array
  MergeSort(arr, l, m, size);

  // Writing to the Pipe-I for Parent to read
  close(fd1[0]);
  write(fd1[1], arr, size*4);
  close(fd1[1]);
  exit(0);
}

// Child Process #2
else if (pid2 == 0) {
  // Sorting the second half of the array
  MergeSort(arr, m + 1, r, size);

  // Writing to the Pipe-II for Parent to read
  close(fd2[0]);
  write(fd2[1], arr, size*4);
  close(fd2[1]);
  exit(0);
```

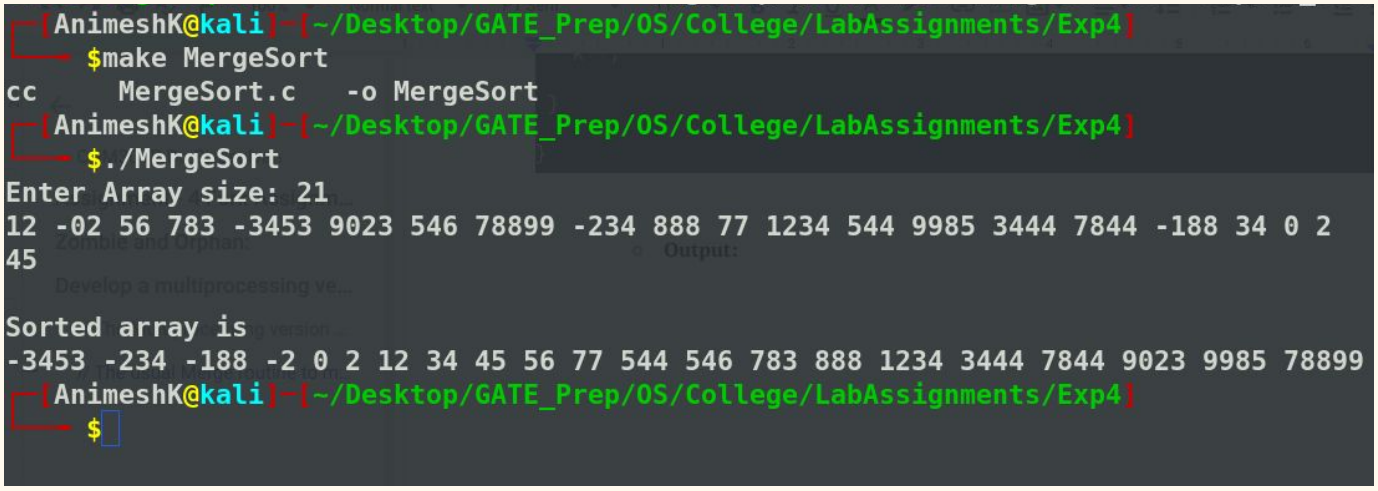
```
  }


  // Parent Process
  else {
    int left[size], right[size];
    while ((wpid = wait(&status)) > 0) // wait till all the children
finish


    // Reading the modified first half of the array from Child-I
    close(fd1[1]);
    read(fd1[0], left, size*4);
    close(fd1[0]);


    // Reading the modified second half of the array from Child-II
    close(fd2[1]);
    read(fd2[0], right, size*4);
    close(fd2[0]);


    /* Copying the relevant data from both 'right' and 'left' part
to the
      original array */
    for (int i = l; i <= m; i++)
      arr[i] = left[i];
```

```
    for (int i = m + 1; i <= r; i++)
      arr[i] = right[i];

    // Merging the sorted halves
    merge(arr, l, m, r);
  }
 }
}

// The usual Merge routine to merge the sorted halves of the array
arr[]
void merge(int arr[], int l, int m, int r)
{
 int i, j, k;
 int n1 = m - l + 1;
 int n2 = r - m;
 int L[n1], R[n2];
 for (i = 0; i < n1; i++)
   L[i] = arr[l + i];
 for (j = 0; j < n2; j++)
   R[j] = arr[m + 1 + j];
 i = 0; // Initial index of first subarray
```

```
j = 0; // Initial index of second subarray
k = l; // Initial index of merged subarray


while (i < n1 && j < n2) {
  if (L[i] <= R[j]) {
    arr[k] = L[i];
    i++;
  }
  else {
    arr[k] = R[j];
    j++;
  }
  k++;
}
// for remaining elements
while (i < n1) {
  arr[k] = L[i];
  i++;
  k++;
}
// for remaining elements
while (j < n2) {
```

```
    arr[k] = R[j];

    j++;

    k++;

  }

}
```

- **Output:**




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

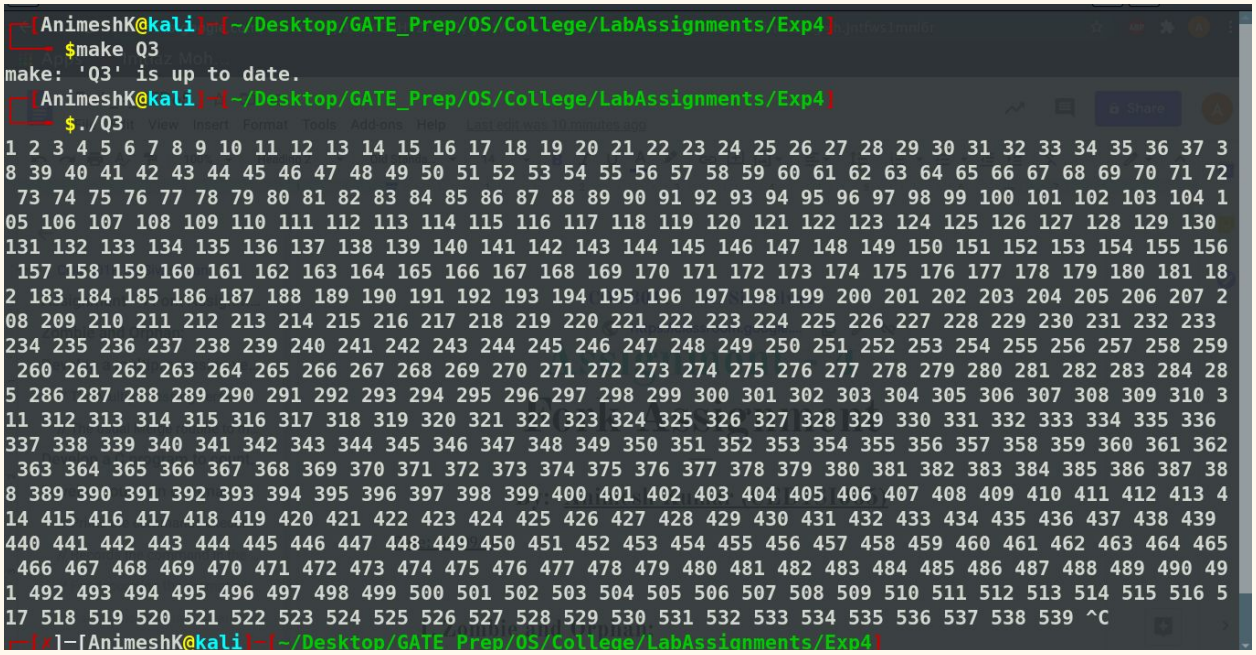
- Algorithm:
  - Run an infinite loop, forking the loop in each iteration while printing the iteration number on stdout.
- C Program:

```
#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>
```

```c
#include <sys/wait.h>
#include <stdlib.h>


int main() {
 pid_t pid;
 int i = 1;
 while(1) {
   pid = fork();
   if(pid < 0) {  // Fork fails after reaching its limits
     printf("MAX no of concurrent process are %d\n",i);
     exit(0);
   }
   if(pid == 0) { // Child Process
     printf("%d ", i);
     i++;
   }
   else {  // Parent Process
     wait(0);
     exit(0);
   }
 }
 return 0;
```

```
}
```

- Output: Created a fork bomb literally. The first time my OS crashed and second time I forced the bomb to terminate before it could reach the limits, not wanting to force the system to shut down from the power button.




# 4. Develop your own command shell. Also extend this to support a history feature.

- **Algorithm:** The below steps are done under an infinite loop: **while(True)**
  - Take input from stdin and if it is not an empty string, then parse it based on the <space> as the delimiter. The Parser() does this job and converts a string into an array of strings args[]. The args[0] holds the command name. We record this argos[0] in the History[][], which acts similar to a queue of fixed buffer size 9. You can only see history for maximum 9 previous commands.
  - if args[0] is
    - exit(): Break the loop

- cd: then use chdir() in this process
- help: print help info

- else we create a child process
  - The child executes the command in a successful execvp() call based on args[] arguments.
  - The parent waits for the child to complete and then does nothing but continue the infinite loop.
- The input exit() breaks the loop and terminates the program.

- **C Program:**

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


char History[9][100];


void PrintHistory(char * args);
void Record(char * cmd);
void Parser(char * buff, char**args);
void RipOffNextLine(char * buff);
```

```
void get_pwd_decorator(char * buff);


int main() {
 while(1) {
   char buff[256];
   char * args[50];
   char curr_path[256];
   int status = 0;
   pid_t wpid;
   get_pwd_decorator(curr_path);
   printf("\n[+] [Animesh@CED18I065]—[%s]\n└──╼ # ", curr_path);
   fgets(buff, sizeof(buff), stdin);
   if(strcmp(buff, "\n") == 0) // In case of empty string
     continue;
   Parser(buff, args);
   Record(args[0]);


   if (args[0][0] == '!')  // History Command
     PrintHistory(args[0]);


   else if (strcmp(args[0], "exit()") == 0) { // For exit()
     printf("Thanks for using.\nCommand Shell Created by ANIMESH
KUMAR CED18I065 2020\n");
```

```
    break;
  }

  else if (strcmp(args[0], "help") == 0) // Special care for 'help'
    printf("1. Use exit() to exit the Command Prompt\n2. The
Pipes(|) do not work\n3. Enter !n, for history, where 0 < n < 10\n");

  else if (strcmp(args[0], "cd") == 0) // Special care for 'cd'
    chdir(args[1]);

  else if (fork() == 0) {   // Child block which executes commands
    if (execvp(args[0], args) < 0)
      printf("[✗]Exec failure: Invalid Input\nEnter 'help' for
help\n");
    exit(0);
  }

  else          // Parent waits for child to complete
    while ((wpid = wait(&status)) > 0);
 } // Infinite loop ends
 return 0;
}
```

```
// Prints the commands executed in history
void PrintHistory(char * args) {
 if ((args[1] > 57) || (args[1] < 49) || (args[2] != '\0')) {
   printf("[✗]History failure: Invalid Input\nEnter 'help' for
help\n");
   return;
 }
 for (int i = 0; i <= args[1] - 49; i++) {
   printf("%s\n", History[i]);
 }
}


// Records the command in the History book
void Record(char * cmd) {
 for (int i = 8; i >= 0; i--) // Making space for the new record
   strcpy(History[i + 1], History[i]);
 strcpy(History[0], cmd);
}



// For decorating the prompt with the 'pwd' output
void get_pwd_decorator(char * buff) {
 int fd[2];
```

```
 pipe(fd);
 if (fork() == 0) { // Child Process
   close(fd[0]);
   dup2(fd[1], 1); // to send the output of 'pwd' to the parent via
pipe
   execlp("pwd", "pwd", NULL);
 }
 else {              // Parent Process
   close(fd[1]);
   read(fd[0], buff, 256);
   close(fd[0]);
   RipOffNextLine(buff); // rip off the trailing '\n'
 }
}


// Parses the 'buff' into an array of strings 'args'
void Parser(char * buff, char**args) {
 int i = 0;
 RipOffNextLine(buff);
 char * token = strtok(buff, " ");
 args[i] = token;
 i++;
 while(1) {
```

```
   token = strtok(NULL, " ");
   if (token == NULL)
     break;
   args[i] = token;
   i++;
 }
 if (strcmp(args[0], "ls") == 0) {
   args[i] = "--color=auto";
   i++;
 }
 args[i] = (char *)NULL;
}


// A simple utility to rip off the trailing '\n' from 'buff'
void RipOffNextLine(char * buff) {
 int i = 0;
 while(buff[i] != '\n')
   i++;
 buff[i] = '\0';
}
```

- **Output:**

```
┌──[AnimeshK@kali]─[~/Desktop/GATE_Prep/OS/College/LabAssignments/Exp4]
└─$make MyShell
make: 'MyShell' is up to date.
┌──[AnimeshK@kali]─[~/Desktop/GATE_Prep/OS/College/LabAssignments/Exp4]
└─$./MyShell

[+] [Animesh@CED18I065]─[/home/AnimeshK/Desktop/GATE_Prep/OS/College/LabAssignments/Exp4]
└──╼ # ls
a.out              CreateMagicSquare.c  Histogram          MatrixMult    MyShell       text_file.txt
class              exp                  Histogram.c        MatrixMult.c  MyShell.c     Zombie_code
cpy2.c             exp.c                MagicSquareCheck   MergeSort     Orphan_code   Zombie_code.c
CreateMagicSquare  hello                MagicSquareCheck.c MergeSort.c   Orphan_code.c

[+] [Animesh@CED18I065]─[/home/AnimeshK/Desktop/GATE_Prep/OS/College/LabAssignments/Exp4]
└──╼ # pwd
/home/AnimeshK/Desktop/GATE_Prep/OS/College/LabAssignments/Exp4

[+] [Animesh@CED18I065]─[/home/AnimeshK/Desktop/GATE_Prep/OS/College/LabAssignments/Exp4]
└──╼ # cd ..

[+] [Animesh@CED18I065]─[/home/AnimeshK/Desktop/GATE_Prep/OS/College/LabAssignments]
└──╼ # cd ..

[+] [Animesh@CED18I065]─[/home/AnimeshK/Desktop/GATE_Prep/OS/College]
└──╼ #
```

```
[+] [Animesh@CED18I065]─[/home/AnimeshK/Desktop/GATE_Prep/OS/College/LabAssignments]
└──╼ # cd Exp4/hello

[+] [Animesh@CED18I065]─[/home/AnimeshK/Desktop/GATE_Prep/OS/College/LabAssignments/Exp4/hello]
└──╼ # ls
bad  good

[+] [Animesh@CED18I065]─[/home/AnimeshK/Desktop/GATE_Prep/OS/College/LabAssignments/Exp4/hello]
└──╼ # cd ..

[+] [Animesh@CED18I065]─[/home/AnimeshK/Desktop/GATE_Prep/OS/College/LabAssignments/Exp4]
└──╼ # echo hello
hello

[+] [Animesh@CED18I065]─[/home/AnimeshK/Desktop/GATE_Prep/OS/College/LabAssignments/Exp4]
└──╼ # ./MergeSort
Enter Array size: 4
12 23 -3 5

Sorted array is
-3 5 12 23

[+] [Animesh@CED18I065]─[/home/AnimeshK/Desktop/GATE_Prep/OS/College/LabAssignments/Exp4]
└──╼ #
```

```
[+] [Animesh@CED18I065]—[/home/AnimeshK/Desktop/GATE_Prep/OS/College/LabAssignments/Exp4]
└──╼ # ls -alh
total 288K
drwxr-xr-x 5 AnimeshK AnimeshK 4.0K Oct  4 20:40 .
drwxr-xr-x 6 AnimeshK AnimeshK 4.0K Oct  3 22:34 ..
-rwxr-xr-x 1 AnimeshK AnimeshK  20K Oct  4 11:17 a.out
drwxr-xr-x 2 AnimeshK AnimeshK 4.0K Oct  1 11:33 class
-rw-r--r-- 1 AnimeshK AnimeshK  473 Oct  4 10:11 cpy2.c
-rwxr-xr-x 1 AnimeshK AnimeshK  22K Oct  3 11:49 CreateMagicSquare
-rw-r--r-- 1 AnimeshK AnimeshK 6.1K Oct  3 11:49 CreateMagicSquare.c
-rwxr-xr-x 1 AnimeshK AnimeshK  17K Oct  4 11:33 exp
-rw-r--r-- 1 AnimeshK AnimeshK  800 Oct  4 11:33 exp.c
drwxr-xr-x 4 AnimeshK AnimeshK 4.0K Sep 26 11:14 hello
-rwxr-xr-x 1 AnimeshK AnimeshK  21K Sep 27 21:15 Histogram
-rwxrwxrwx 1 AnimeshK AnimeshK 8.4K Sep 27 21:14 Histogram.c
-rwxr-xr-x 1 AnimeshK AnimeshK  17K Oct  3 09:54 MagicSquareCheck
-rw-r--r-- 1 AnimeshK AnimeshK 6.0K Oct  2 11:50 MagicSquareCheck.c
-rwxr-xr-x 1 AnimeshK AnimeshK  17K Sep 27 21:59 MatrixMult
-rw-r--r-- 1 AnimeshK AnimeshK 2.1K Sep 27 21:59 MatrixMult.c
-rwxr-xr-x 1 AnimeshK AnimeshK  17K Oct  4 20:40 MergeSort
-rw-r--r-- 1 AnimeshK AnimeshK 3.1K Oct  4 20:35 MergeSort.c
-rwxr-xr-x 1 AnimeshK AnimeshK  18K Oct  4 12:33 MyShell
-rw-r--r-- 1 AnimeshK AnimeshK 3.1K Oct  4 12:32 MyShell.c
-rwxr-xr-x 1 AnimeshK AnimeshK  17K Sep 24 09:55 Orphan_code
-rw-r--r-- 1 AnimeshK AnimeshK  528 Sep 24 09:54 Orphan_code.c
```

```
[+] [Animesh@CED18I065]—[/home/AnimeshK/Desktop/GATE_Prep/OS/College/LabAssignments/Exp4]
└──╼ # help
1. Use exit() to exit the Command Prompt
2. The Pipes(|) do not work
3. Enter !n, for history, where 0 < n < 10

[+] [Animesh@CED18I065]—[/home/AnimeshK/Desktop/GATE_Prep/OS/College/LabAssignments/Exp4]
└──╼ # !8
!8
help
clear
ls
clear
./MergeSort
echo
cd

[+] [Animesh@CED18I065]—[/home/AnimeshK/Desktop/GATE_Prep/OS/College/LabAssignments/Exp4]
└──╼ # exit()
Thanks for using.
Command Shell Created by ANIMESH KUMAR CED18I065 2020
┌──[AnimeshK@kali]-[~/Desktop/GATE_Prep/OS/College/LabAssignments/Exp4]
└─╼ $
```

**In this question I learnt a lot. Thanks a lot.**

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

- **Algorithm:** Here I interpreted character as the alphabet in a lower case. The process is done over the `text_file.txt.(by default)`
  - Open the file in read mode
  - fork() 25 new Child processes.
  - Each of the child processes count the frequency of a given letter. Prints the frequency of that letter in a graphical manner(*kind of graphical : )*);
  - The parent process waits for all children and then counts the frequency of 'z' in the input file. Prints the frequency of 'z'.
  - Terminate successfully
- **C Program:**

```c
#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<sys/types.h>

#include<string.h>

#include<sys/wait.h>

#include<time.h>


void CreateHistogram(char * filename);


int main  () {
```

```
 char * filename = "text_file.txt";

 CreateHistogram(filename);

 return 0;

}


void CreateHistogram(char * filename) {

 __pid_t pid1, pid2, pid3, pid4, pid5, pid6, pid7, pid8, pid9, pid10,
pid11, pid12, pid13, pid14, pid15, pid16, pid17, pid18, pid19, pid20,
pid21, pid22, pid23, pid24, pid25, wpid;

 int status = 0;


 // Opening the text file to read the text input.

 FILE * fptr = fopen(filename, "r");


/* We need to fork 25 new child processes, each of them counting the
frequency

 of a single alphabet(smaller case)*/

 ((pid1 = fork()) && (pid2 = fork()) && (pid3 = fork()) && (pid4 =
fork()) && (pid5 = fork()) && (pid6 = fork()) && (pid7 = fork()) &&
(pid8 = fork()) && (pid9 = fork()) && (pid10 = fork()) && (pid11 =
fork()) && (pid12 = fork()) && (pid13 = fork()) && (pid14 = fork())
&& (pid15 = fork()) && (pid16 = fork()) && (pid17 = fork()) && (pid18
= fork()) && (pid19 = fork()) && (pid20 = fork()) && (pid21 = fork())
&& (pid22 = fork()) && (pid23 = fork()) && (pid24 = fork()) && (pid25
= fork()));
```

```
// Computes the frequency of a's
 if (pid1 == 0) {
   int count = 0;
   printf("a: ");
   rewind(fptr);
   char ch = fgetc(fptr);
   while (ch != EOF) {
     if (ch == 'a') {
       printf("[_]");
       count++;
     }
     ch = fgetc(fptr);
   }
   printf("     (%d)\n", count);
 }
// Computes the frequency of b's and so on ...
 else if (pid2 == 0) {
   int count = 0;
   printf("b: ");
   rewind(fptr);
   char ch = fgetc(fptr);
   while (ch != EOF) {
```

```
    if (ch == 'b') {
      printf("[_]");
      count++;
    }
    ch = fgetc(fptr);
  }
  printf("      (%d)\n", count);
}
else if (pid3 == 0) {
  int count = 0;
  rewind(fptr);
  printf("c: ");
  char ch = fgetc(fptr);
  while (ch != EOF) {
    if (ch == 'c') {
      printf("[_]");
      count++;
    }
    ch = fgetc(fptr);
  }
  printf("      (%d)\n", count);
}
```

```
else if (pid4 == 0) {
  int count = 0;
  rewind(fptr);
  printf("d: ");
  char ch = fgetc(fptr);
  while (ch != EOF) {
    if (ch == 'd') {
      printf("[_]");
      count++;
    }
    ch = fgetc(fptr);
  }
  printf("     (%d)\n", count);
}
else if (pid5 == 0) {
  int count = 0;
  rewind(fptr);
  printf("e: ");
  char ch = fgetc(fptr);
  while (ch != EOF) {
    if (ch == 'e') {
      printf("[_]");
```

```
      count++;
    }
    ch = fgetc(fptr);
  }
  printf("     (%d)\n", count);
}
else if (pid6 == 0) {
  int count = 0;
  rewind(fptr);
  printf("f: ");
  char ch = fgetc(fptr);
  while (ch != EOF) {
    if (ch == 'f') {
      printf("[_]");
      count++;
    }
    ch = fgetc(fptr);
  }
  printf("     (%d)\n", count);
}
else if (pid7 == 0) {
  int count = 0;
```

```
  rewind(fptr);
  printf("g: ");
  char ch = fgetc(fptr);
  while (ch != EOF) {
    if (ch == 'g') {
      printf("[_]");
      count++;
    }
    ch = fgetc(fptr);
  }
  printf("     (%d)\n", count);
}
else if (pid8 == 0) {
  int count = 0;
  rewind(fptr);
  printf("h: ");
  char ch = fgetc(fptr);
  while (ch != EOF) {
    if (ch == 'h') {
      printf("[_]");
      count++;
    }
```

```
    ch = fgetc(fptr);
  }
  printf("     (%d)\n", count);
}
else if (pid9 == 0) {
  int count = 0;
  rewind(fptr);
  printf("i: ");
  char ch = fgetc(fptr);
  while (ch != EOF) {
    if (ch == 'i') {
      printf("[_]");
      count++;
    }
    ch = fgetc(fptr);
  }
  printf("     (%d)\n", count);
}
else if (pid10 == 0) {
  int count = 0;
  rewind(fptr);
  printf("j: ");
```

```
  char ch = fgetc(fptr);
  while (ch != EOF) {
    if (ch == 'j') {
      printf("[_]");
      count++;
    }
    ch = fgetc(fptr);
  }
  printf("     (%d)\n", count);
}
else if (pid11 == 0) {
  int count = 0;
  rewind(fptr);
  printf("k: ");
  char ch = fgetc(fptr);
  while (ch != EOF) {
    if (ch == 'k') {
      printf("[_]");
      count++;
    }
    ch = fgetc(fptr);
  }
```

```
  printf("      (%d)\n", count);
}
else if (pid12 == 0) {
  int count = 0;
  rewind(fptr);
  printf("l: ");
  char ch = fgetc(fptr);
  while (ch != EOF) {
    if (ch == 'l') {
      printf("[_]");
      count++;
    }
    ch = fgetc(fptr);
  }
  printf("      (%d)\n", count);
}
else if (pid13 == 0) {
  int count = 0;
  rewind(fptr);
  printf("m: ");
  char ch = fgetc(fptr);
  while (ch != EOF) {
```

```
    if (ch == 'm') {
      printf("[_]");
      count++;
    }
    ch = fgetc(fptr);
  }
  printf("      (%d)\n", count);
}
else if (pid14 == 0) {
  int count = 0;
  rewind(fptr);
  printf("n: ");
  char ch = fgetc(fptr);
  while (ch != EOF) {
    if (ch == 'n') {
      printf("[_]");
      count++;
    }
    ch = fgetc(fptr);
  }
  printf("      (%d)\n", count);
}
```

```
else if (pid15 == 0) {
  int count = 0;
  rewind(fptr);
  printf("o: ");
  char ch = fgetc(fptr);
  while (ch != EOF) {
    if (ch == 'o') {
      printf("[_]");
      count++;
    }
    ch = fgetc(fptr);
  }
  printf("     (%d)\n", count);
}
else if (pid16 == 0) {
  int count = 0;
  rewind(fptr);
  printf("p: ");
  char ch = fgetc(fptr);
  while (ch != EOF) {
    if (ch == 'p') {
      printf("[_]");
```

```
      count++;
    }
    ch = fgetc(fptr);
  }
  printf("     (%d)\n", count);
}
else if (pid17 == 0) {
  int count = 0;
  rewind(fptr);
  printf("q: ");
  char ch = fgetc(fptr);
  while (ch != EOF) {
    if (ch == 'q') {
      printf("[_]");
      count++;
    }
    ch = fgetc(fptr);
  }
  printf("     (%d)\n", count);
}
else if (pid18 == 0) {
  int count = 0;
```

```
  rewind(fptr);
  printf("r: ");
  char ch = fgetc(fptr);
  while (ch != EOF) {
    if (ch == 'r') {
      printf("[_]");
      count++;
    }
    ch = fgetc(fptr);
  }
  printf("     (%d)\n", count);
}
else if (pid19 == 0) {
  int count = 0;
  rewind(fptr);
  printf("s: ");
  char ch = fgetc(fptr);
  while (ch != EOF) {
    if (ch == 's') {
      printf("[_]");
      count++;
    }
```

```
    ch = fgetc(fptr);
  }
  printf("      (%d)\n", count);
}
else if (pid20 == 0) {
  int count = 0;
  rewind(fptr);
  printf("t: ");
  char ch = fgetc(fptr);
  while (ch != EOF) {
    if (ch == 't') {
      printf("[_]");
      count++;
    }
    ch = fgetc(fptr);
  }
  printf("      (%d)\n", count);
}
else if (pid21 == 0) {
  int count = 0;
  rewind(fptr);
  printf("u: ");
```

```
  char ch = fgetc(fptr);
  while (ch != EOF) {
    if (ch == 'u') {
      printf("[_]");
      count++;
    }
    ch = fgetc(fptr);
  }
  printf("     (%d)\n", count);
}
else if (pid22 == 0) {
  int count = 0;
  rewind(fptr);
  printf("v: ");
  char ch = fgetc(fptr);
  while (ch != EOF) {
    if (ch == 'v') {
      printf("[_]");
      count++;
    }
    ch = fgetc(fptr);
  }
```

```
  printf("      (%d)\n", count);
}
else if (pid23 == 0) {
  int count = 0;
  rewind(fptr);
  printf("w: ");
  char ch = fgetc(fptr);
  while (ch != EOF) {
    if (ch == 'w') {
      printf("[_]");
      count++;
    }
    ch = fgetc(fptr);
  }
  printf("      (%d)\n", count);
}
else if (pid24 == 0) {
  int count = 0;
  rewind(fptr);
  printf("x: ");
  char ch = fgetc(fptr);
  while (ch != EOF) {
```

```
    if (ch == 'x') {
      printf("[_]");
      count++;
    }
    ch = fgetc(fptr);
  }
  printf("      (%d)\n", count);
}
else if (pid25 == 0) {
  int count = 0;
  rewind(fptr);
  printf("y: ");
  char ch = fgetc(fptr);
  while (ch != EOF) {
    if (ch == 'y') {
      printf("[_]");
      count++;
    }
    ch = fgetc(fptr);
  }
  printf("      (%d)\n", count);
}
```

```
else {
  while ((wpid = wait(&status)) > 0);
  int count = 0;
  rewind(fptr);
  printf("z: ");
  char ch = fgetc(fptr);
  while (ch != EOF) {
    if (ch == 'z') {
      printf("[_]");
      count++;
    }
    ch = fgetc(fptr);
  }
  printf("     (%d)\n", count);
  printf("\n**********The Histogram of Occurences of all the Characters*********\n");
 }
 }
```

- **Output:**

```
┌──[AnimeshK@kali]─[~/Desktop/GATE_Prep/OS/College/LabAssignments/Exp4]
└─$make Histogram
make: 'Histogram' is up to date.
┌──[AnimeshK@kali]─[~/Desktop/GATE_Prep/OS/College/LabAssignments/Exp4]
└─$./Histogram
a: [_][_][_][_][_][_][_][_][_][_][_][_][_][_][_][_][_][_][_][_][_][_][_][_][_][_][_][_]     (26)
b:      (0)
c:      (0)
d:      (0)
e:      (0)
f:      (0)
g:      (0)
h: [_][_][_][_][_][_][_][_][_][_][_][_][_][_][_][_][_][_][_]     (19)
i:      (0)
j:      (0)
k: [_][_][_][_][_][_][_][_]     (8)
l:      (0)
m:      (0)
n:      (0)
o: [_][_]     (2)
p:      (0)
q:      (0)
r:      (0)
s:      (0)
t:      (0)
v:      (0)
```

```
f:      (0)
g:      (0)
h: [_][_][_][_][_][_][_][_][_][_][_][_][_][_][_][_][_][_][_]     (19)
i:      (0)
j:      (0)
k: [_][_][_][_][_][_][_][_]     (8)
l:      (0)
m:      (0)
n:      (0)
o: [_][_]     (2)
p:      (0)
q:      (0)
r:      (0)
s:      (0)
t:      (0)
v:      (0)
u: [_][_][_][_][_][_][_][_][_][_][_][_][_][_][_][_][_][_][_][_][_][_][_][_]     (24)
w:      (0)
x:      (0)
y:      (0)
z: [_][_]     (2)

**********The Histogram of Occurences of all the Characters**********
┌──[AnimeshK@kali]─[~/Desktop/GATE_Prep/OS/College/LabAssignments/Exp4]
└─$
```

# 6. Develop a multiprocessing version of matrix multiplication.

- **Algorithm:** The algorithm is designed only for 3 X 3 matrices. It takes two 3 X 3 matrices as input and prints out the Product matrix.

  **Component() function**

  - Calculate the ith row of first matrix dot product with jth column of the second matrix.
  - Return this integer.

  **Multiply() function**

  - For every (i, j) combination: ith row of first matrix and jth column of the second matrix fork() a child process, which
    - calculates the Component() and sends the product to the parent via pipe.
    - Terminate this child.
    - The parent process receives the Component values via pipes from respective children and modifies the Product[][] matrix.

  Print the Product[][] Matrix and terminate the parent process.

- **C Program:**

```
#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<sys/types.h>

#include<string.h>

#include<sys/wait.h>

#include<time.h>
```

```
void take_input(int arr[][3]);

void print_mat(int arr[][3]);

void Multiply(int Product[][3], int First[][3], int Second[][3], int
size);

int Component(int ptr1, int ptr2, int First[][3], int Second[][3],
int size);


int main() {

 int First[3][3];

 int Second[3][3];

 int Product[3][3];

 printf("Enter the First matrix row by row: \n");

 take_input(First);

 printf("Enter the Second matrix row by row: \n");

 take_input(Second);

 Multiply(Product, First, Second, 3);

 printf("The Product matrix is as below: \n");

 print_mat(Product);

 return 0;

}
```

```
// A Multiprocessing implementation of Matrix Multiplication
Algorithm

void Multiply(int Product[][3], int First[][3], int Second[][3], int
size) {

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

   for (int j = 0; j < size; j++) {

     int fd[2];           // File Descriptor Array

     pipe(fd);

     if (fork() == 0) {  // Child Process

       int prod = Component(i, j, First, Second, size);


       // Writing the Component Product value to the pipe

       close(fd[0]);

       write(fd[1], &prod, 10);

       close(fd[1]);

       exit(0);

     }

     else {              // Parent Process

       // Reading the Component Product value from the pipe

       close(fd[1]);

       read(fd[0], &Product[i][j], 10); // Updating the Product
matrix

       close(fd[0]);
```

```
    }
  }
 }
}


// Component Product Calculator
int Component(int ptr1, int ptr2, int First[][3], int Second[][3],
int size) {
 int sum = 0;
 for (int i = 0; i < size; i++)
   sum += (First[ptr1][i] * Second[i][ptr2]);
 return sum;
}


// A simple utility to take the Matrix as input
void take_input(int arr[][3]) {
 for (int i = 0; i < 3; i++) {
   for (int j = 0; j < 3; j++)
     scanf("%d", &arr[i][j]);
 }
}


// A simple utility to print out the matrix
```

```c
void print_mat(int arr[][3]) {

 for (int i = 0; i < 3; i++) {

   for (int j = 0; j < 3; j++)

     printf("%d ", arr[i][j]);

   printf("\n");

 }

 printf("\n");

}
```

- **Output:**

```
┌──[AnimeshK@kali]─[~/Desktop/GATE_Prep/OS/College/LabAssignments/Exp4]
└─$make MatrixMult
make: 'MatrixMult' is up to date.
┌──[AnimeshK@kali]─[~/Desktop/GATE_Prep/OS/College/LabAssignments/Exp4]
└─$./MatrixMult
Enter the First matrix row by row:
1 3 4
7 2 4
-1 3 9
Enter the Second matrix row by row:
9 0 656
-34 9 41
90 -4 56
The Product matrix is as below:
267 11 1003
355 2 4898
699 -9 -29

┌──[AnimeshK@kali]─[~/Desktop/GATE_Prep/OS/College/LabAssignments/Exp4]
└─$
```

# 7. Develop a parallelized application to check for if a user input square matrix is a magic square or not.

- **Algorithm:** Input the size of the square, and then input the square. The Program will output the magic constant if the input is a magic square otherwise it will let you know. The pseudo code of is_magic() function is:
  - Fork 4 new child processes.
  - The First Child Process calls RowSumHandler() which creates one child process for each row, each such child process
    - Calculates the sum of that row elements and sends the sum to the parent via pipes.
    - The parent compares the sum values got from the child processes associated with each row.
      a. If all the sums are same, it returns this magic sum
      b. else return -1 as indication of non-magic square
    - This returned value is sent to the original parent via pipes
  - The second child Process calls ColumnSumHandler() which creates one child process for each column, each such child process
    - Calculates the sum of that column elements and sends the sum to the parent via pipes.
    - The parent compares the sum values got from the child processes associated with each column.
      a. If all the sums are same, it returns this magic sum
      b. else return -1 as indication of non-magic square
    - This returned value is sent to the original parent via pipes
  - The third child process calculates the LEFT diagonal sum and sends it to the original parent via pipes.

- The fourth child process calculates the RIGHT diagonal sum and sends it to the original parent via pipes.
- The parent process receives four sum values via pipes from all these 4 children.
    - If these sum values are equal: It prints the sum and terminates the code. The square is a magic square.
    - else: The square is not a magic square. terminates.

- **C Program:**

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


#define LEFT 0
#define RIGHT 1


int size = 0;


void take_input(int arr[][size]);
void print_mat(int arr[][size] );
int RowSumHandler(int Matrix[][size]);
```

```c
int ColumnSumHandler(int Matrix[][size]);

int get_column_sum(int Matrix[][size], int ptr);

int get_row_sum(int Matrix[][size], int ptr);

int DiagonalSum(int flag, int Matrix[][size]);

int is_magic(int Matrix[][size]);


int main() {

 printf("Enter the Size N of N X N Square: ");

 scanf("%d", &size);

 int Matrix[size][size];

 printf("Enter the Matrix row by row: \n");

 take_input(Matrix);

  int magic_const = is_magic(Matrix);

 if(magic_const > 0)

   printf(":) The Matrix IS a Magic Square with Magic Constant %d
\n", magic_const);

 else

   printf("The Matrix IS NOT a Magic Square :( \n");

 return 0;

}


// This function calculates the magic constant of the square(If it is
Magic Sq)
```

```
int is_magic(int Matrix[][size]) {
 pid_t pid1, pid2, pid3, pid4;
 int fd1[2], fd2[2], fd3[2], fd4[2];
 pipe(fd1);
 pipe(fd2);
 pipe(fd3);
 pipe(fd4);
 // Forking 4 child processes
 ((pid1 = fork()) && (pid2 = fork()) && (pid3 = fork()) && (pid4 =
fork()));
 if (pid1 == 0)  {    // Child 1 block
   int sum = RowSumHandler(Matrix);
   // Send the 'sum' to parent process via pipe
   close(fd1[0]);
   write(fd1[1], &sum, 4);
   close(fd1[1]);
   // Succesful termination
   exit(0);
 }
 else if (pid2 == 0)  { // Child 2 block
   int sum = ColumnSumHandler(Matrix);
   // Send the 'sum' to parent process via pipe
   close(fd2[0]);
```

```
  write(fd2[1], &sum, 4);
  close(fd2[1]);
  // Succesful termination
  exit(0);
}
else if (pid3 == 0)  {  // Child 3 block
  int sum = DiagonalSum(RIGHT, Matrix);
  // Send the 'sum' to parent process via pipe
  close(fd3[0]);
  write(fd3[1], &sum, 4);
  close(fd3[1]);
  // Succesful termination
  exit(0);
}
else if (pid4 == 0)  {   // Child 4 block
  int sum = DiagonalSum(RIGHT, Matrix);
  // Send the 'sum' to parent process via pipe
  close(fd4[0]);
  write(fd4[1], &sum, 4);
  close(fd4[1]);
  // Succesful termination
  exit(0);
```

```
}
else {   // The Paren Block
  int row_sum, column_sum, diagonal_1_sum, diagonal_2_sum;
   // Recieving the Row Sum from First Child
  close(fd1[1]);
  read(fd1[0], &row_sum, 4);
  close(fd1[0]);

  // Recieving the Column Sum from Second Child
  close(fd2[1]);
  read(fd2[0], &column_sum, 4);
  close(fd2[0]);

  // Recieving the Sum of Main diagonal(LEFT) from Third Child
  close(fd3[1]);
  read(fd3[0], &diagonal_1_sum, 4);
  close(fd3[0]);

  // Recieving the Sum of Other diagonal(RIGHT)from Fourth Child
  close(fd4[1]);
  read(fd4[0], &diagonal_2_sum, 4);
  close(fd4[0]);
```

```
   // Comparing the Sum values
   if ((row_sum == column_sum) && (column_sum == diagonal_1_sum) &&
(diagonal_1_sum == diagonal_2_sum))
     return row_sum; // Returning the Magic Constant of this Magic
Square
   return -1;  // Returning the signal of not being a magic square
 }
}

// This function calculates the Diagonal Sum(RIGHT or LEFT) based on
the flag value
int DiagonalSum(int flag, int Matrix[][size]) {
 int sum = 0;
 for (int i = 0; i < size; i++) {
   if (flag == LEFT)  // In case of main diagonal
     sum += Matrix[i][i];
   else               // In case of other diagonal
     sum += Matrix[size - 1 - i][i];
 }
 return sum;
}
```

```
// This function compares the sum of all the rows and returns final
row_sum(if any)
int RowSumHandler(int Matrix[][size]) {
 int prev_sum = 0;
 int sum;
 for(int i = 0; i < size; i++) {
   int fd[2];
   pipe(fd);
   pid_t pid = fork(); // Forking the Child process to calculate the
Row Sum
   if (pid == 0) {
     sum = get_row_sum(Matrix, i); // Calculates Row sum of ith row
of the Matrix
     // Sending the row sum to the parent process
     close(fd[0]);
     write(fd[1], &sum, 4);
     close(fd[1]);
     exit(0); // successful termination
   }
   else {
     // Receiving the Row Sum from child process
     close(fd[1]);
     read(fd[0], &sum, 4);
```

```
    close(fd[0]);
    if ((prev_sum != 0) && (prev_sum != sum)) // When the
consecutive sums don't match
      return -1;
    prev_sum = sum;
   }
 }
 return prev_sum;
}


// This function compares the sum of all the columns and returns
final column_sum(if any)
int ColumnSumHandler(int Matrix[][size]) {
 int prev_sum = 0;
 int sum;
 for(int i = 0; i < size; i++) {
   int fd[2];
   pipe(fd);
   pid_t pid = fork(); // Forking the Child process to calculate the
Column Sum
   if (pid == 0) {
    sum = get_column_sum(Matrix, i); // Calculates Column sum of ith
Column of the Matrix
    // Sending the column sum to the parent process
```

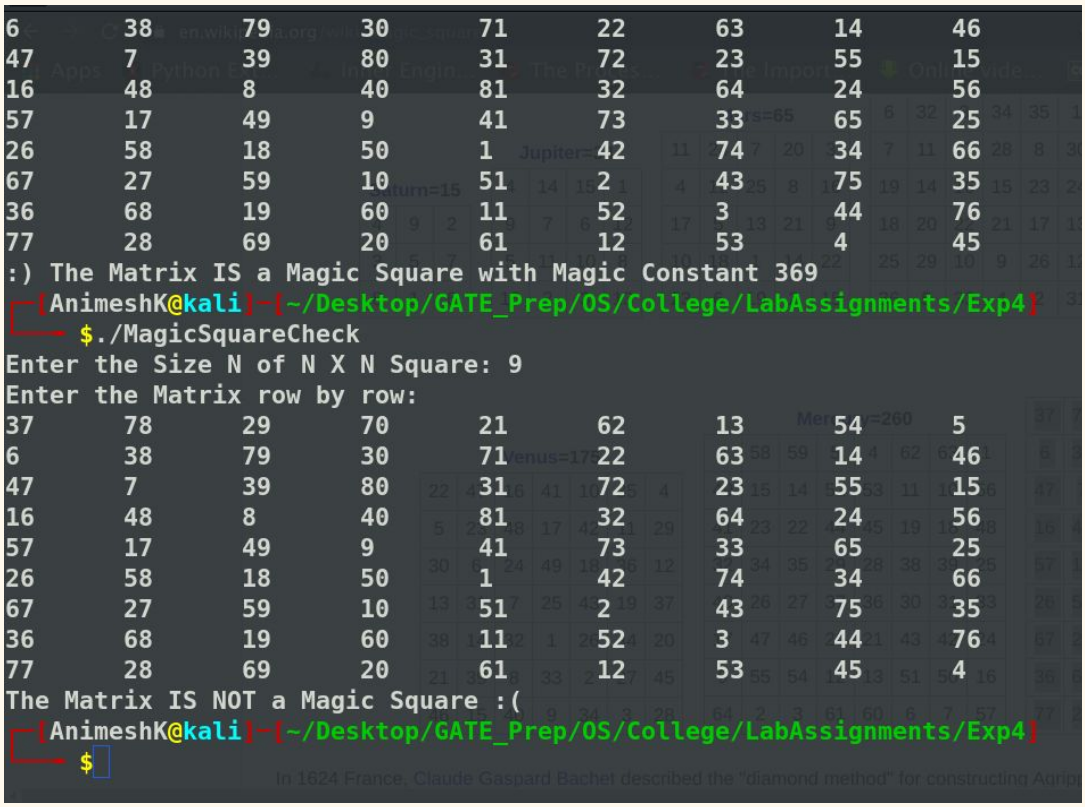
```
    close(fd[0]);
    write(fd[1], &sum, 4);
    close(fd[1]);
    exit(0); // successful termination
   }
   else {
    // Receiving the Column Sum from child process
    close(fd[1]);
    read(fd[0], &sum, 4);
    close(fd[0]);
    if ((prev_sum != 0) && (prev_sum != sum)) // When the
consecutive sums don't match
      return -1;
    prev_sum = sum;
   }
 }
 return prev_sum;
}


// Returns the sum of ptr'th row in the Matrix[][size]
int get_row_sum(int Matrix[][size], int ptr) {
 int sum = 0;
 for (int i = 0; i < size; i++)
```

```
   sum += Matrix[ptr][i];
 return sum;
}


// Returns the sum of ptr'th column in the Matrix[][size]
int get_column_sum(int Matrix[][size], int ptr) {
 int sum = 0;
 for (int i = 0; i < size; i++)
   sum += Matrix[i][ptr];
 return sum;
}


// A simple utility to take the Matrix as input
void take_input(int arr[][size]) {
 for (int i = 0; i < size; i++) {
   for (int j = 0; j < size; j++)
     scanf("%d", &arr[i][j]);
 }
}


// A simple utility to print out the matrix
void print_mat(int arr[][size]) {
```

```
 for (int i = 0; i < size; i++) {

   for (int j = 0; j < size; j++)

     printf("%d ", arr[i][j]);

   printf("\n");

 }

 printf("\n");

}
```

- **Output:**

```
┌──[AnimeshK@kali]-[~/Desktop/GATE_Prep/OS/College/LabAssignments/Exp4]
└─$make MagicSquareCheck
make: 'MagicSquareCheck' is up to date.
┌──[AnimeshK@kali]-[~/Desktop/GATE_Prep/OS/College/LabAssignments/Exp4]
└─$./MagicSquareCheck
Enter the Size N of N X N Square: 9
Enter the Matrix row by row:
37      78      29      70      21      62      13      54      5
6       38      79      30      71      22      63      14      46
47      7       39      80      31      72      23      55      15
16      48      8       40      81      32      64      24      56
57      17      49      9       41      73      33      65      25
26      58      18      50      1       42      74      34      66
67      27      59      10      51      2       43      75      35
36      68      19      60      11      52      3       44      76
77      28      69      20      61      12      53      4       45
:) The Matrix IS a Magic Square with Magic Constant 369
┌──[AnimeshK@kali]-[~/Desktop/GATE_Prep/OS/College/LabAssignments/Exp4]
└─$./MagicSquareCheck
Enter the Size N of N X N Square: 9
Enter the Matrix row by row:
37      78      29      70      21      62      13      54      5
6       38      79      30      71      22      63      14      46
47      7       39      80      31      72      23      55      15
16      48      8       40      81      32      64      24      56
57      17      49      9       41      73      33      65      25
```




# 8. Extend the above to also support magic square generation

- **Algorithm:** I created a new program altogether, I did not extend the above code itself. This program accepts the size of the magic square. **<u>The code is written only for the Doubly even order and Odd order magic squares</u>**. It doesn't work for Singly even order magic square(4*n + 2 type).

  **For doubly even order,** four child processes are forked.

  - All four processes, apply the algorithm over one of the four quarter squares present at the four corners of the given square. Each of them sends the modified matrix to the parent process via pipes and terminate successfully
  - The parent process receives the modified matrix via pipes.

- Then it applies the algorithm over the central square.
- Then it copies the relevant corner squares to the main square and terminates the process after printing the generated magic square.

**For odd order**, the algorithm itself is inherently serial in nature, therefore parallelization is not possible here.

- We start from 1 and go upto n^2 and in each iteration
  - Take a normal diagonal move
  - While you don't reach an unoccupied cell, keep taking a special 'horse move'
  - Fill the unoccupied cell
- Terminate the program after printing the generated magic square.

(For detailed description of these moves, look at the code!)

- **C Program:**

```
#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<sys/types.h>

#include<string.h>

#include<sys/wait.h>

#include<time.h>


#define LEFT 0

#define RIGHT 1
```

```
int size = 0;


void CreateMagic(int Matrix[][size]);

void print_mat(int arr[][size] );


void CreateMagicEven(int Matrix[][size]);

void InitializeMatrix(int Matrix[][size]);

void ModifyQuarter(int Matrix[][size], int ptr, int start, int end);

void CopyMatrix(int Source[][size], int Dest[][size], int start_i,
int start_j);


void CreateMagicOdd(int Matrix[][size]);

int wrap(int i);

void Wrapper(int *i, int *j);

void HorseMove(int *i, int *j);

void NormalMove(int *i, int *j);



int main() {

 printf("Kindly Enter either Odd Size or Doubly Even Size. Singly
even order(4*n + 2 type) IS NOT ALLOWED.\n\nEnter the Size N of N X N
Magic Square: ");

 scanf("%d", &size);
```

```
 int Matrix[size][size];
 memset(Matrix, 0, sizeof(Matrix));
 CreateMagic(Matrix);
 printf("The Magic Square row by row: \n");
 print_mat(Matrix);
 return 0;
}


// Creates the Magic Square
void CreateMagic(int Matrix[][size]) {
 if (size % 2 != 0)  // Odd Order Magic Square
   CreateMagicOdd(Matrix);
 else if (size % 4 == 0) // Doubly Even Order Magic Square
   CreateMagicEven(Matrix);
 else {
   printf("[X]Invalid Input: Singly Even Order Detected
:(\nExiting...\n");
   exit(1);
 }
}


/************** The Doubly even order Magic Square Calculations
************/
```

```
// Creates the Magic Square of Doubly even Order
void CreateMagicEven(int Matrix[][size]) {
 InitializeMatrix(Matrix);
 pid_t pid1, pid2, pid3, pid4;
 int fd1[2], fd2[2], fd3[2], fd4[2];
 pipe(fd1);
 pipe(fd2);
 pipe(fd3);
 pipe(fd4);

 // Forking 4 child processes to establish parellelism
 ((pid1 = fork()) && (pid2 = fork()) && (pid3 = fork()) && (pid4 =
fork()));
  if (pid1 == 0) {
   for (int i = 0; i < size / 4; i++)  // Modify Top left quarter
square
     ModifyQuarter(Matrix, i, 0, size / 4);
   close(fd1[0]);
   write(fd1[1], Matrix, size*size*4);  // Sending the modified
Matrix to Parent
   close(fd1[1]);
   exit(0);
```

```
 }
 else if (pid2 == 0) {
   for (int i = 0; i < size / 4; i++)  // Modify Bottom Left quarter
square
     ModifyQuarter(Matrix, i, 3 * size / 4, size);
   close(fd2[0]);
   write(fd2[1], Matrix, size*size*4); // Sending the modified Matrix
to Parent
   close(fd2[1]);
   exit(0);
 }
 else if (pid3 == 0) {
   for (int i = 3 * size / 4; i < size; i++) // Modify Top Right
quarter square
     ModifyQuarter(Matrix, i, 0, size / 4);
   close(fd3[0]);
   write(fd3[1], Matrix, size*size*4); // Sending the modified Matrix
to Parent
   close(fd3[1]);
   exit(0);
 }
 else if (pid4 == 0) {
   for (int i = 3 * size / 4; i < size; i++) // Modify Bottom Right
quarter square
```

```
    ModifyQuarter(Matrix, i, 3 * size / 4, size);
   close(fd4[0]);
   write(fd4[1], Matrix, size*size*4); // Sending the modified Matrix
to Parent
   close(fd4[1]);
   exit(0);
 }
 else {
   int TopLeft[size][size], BottomLeft[size][size],
TopRight[size][size], BottomRight[size][size];
   // Reading the Modified Quarter Square segments from the 4
children
   close(fd1[1]);
   read(fd1[0], TopLeft, size * size * 4);
   close(fd1[0]);

   close(fd2[1]);
   read(fd2[0], BottomLeft, size * size * 4);
   close(fd2[0]);

   close(fd3[1]);
   read(fd3[0], TopRight, size * size * 4);
   close(fd3[0]);
```

```
   close(fd4[1]);
   read(fd4[0], BottomRight, size * size * 4);
   close(fd4[0]);

   for (int i = size / 4; i < 3 * size / 4; i++) // Modify Central
semi square
     ModifyQuarter(Matrix, i, size / 4, 3 * size / 4);

   // Copying the modified Quarter Square Segments from children to
the Main Matrix
   CopyMatrix(TopLeft, Matrix, 0, 0);
   CopyMatrix(TopRight, Matrix, 0, 3 * size / 4);
   CopyMatrix(BottomLeft, Matrix, 3 * size / 4, 0);
   CopyMatrix(BottomRight, Matrix, 3 * size / 4, 3 * size / 4);
 }
}

// Copies the passed portions of Dest[][] to Source[][]
void CopyMatrix(int Source[][size], int Dest[][size], int start_i,
int start_j) {
 for (int i = start_i; i < start_i + (size / 4); i++) {
   for (int j = start_j; j < start_j + (size / 4); j++)
```

```
     Dest[i][j] = Source[i][j];
 }
}


// Initializes the Doubly even order matrix
void InitializeMatrix(int Matrix[][size]) {
 int k = 1;
 for(int i = 0; i < size; i++) {
   for(int j = 0; j < size; j++) {
     Matrix[i][j] = k;
     k++;
   }
 }
}


// Modifies the matrix based upon the algorithm
void ModifyQuarter(int Matrix[][size], int ptr, int start, int end)
{
 for(int i = start; i < end; i++)
   Matrix[i][ptr] = (size*size + 1) - Matrix[i][ptr];
}
```

```
/*****************************************************************
*******/



/********************* The Odd order Magic Square Calculations
*************/


// Creates the Magic Square of Doubly even Order
void CreateMagicOdd(int Matrix[][size]) {
 int max_num = size*size;
 int x = size / 2;
 int y = size - 1;
 int i = 1;
 Matrix[x][y] = i;
 i++;
 while(i <= max_num) {
   NormalMove(&x, &y);
   while(Matrix[x][y] != 0) // Do horsemoves until reach an empty
cell
     HorseMove(&x, &y);
   Matrix[x][y] = i;
   i++;
 }
```

```
}


// A normal move in case of next cell is not pre-occupied cell
void NormalMove(int *i, int *j) {
 *i = *i - 1;
 *j = *j + 1;
 Wrapper(i, j);
}


// A horse move in case of already occupied next cell
void HorseMove(int *i, int *j) {
 *i = *i + 1;
 *j = *j - 2;
 Wrapper(i, j);
}


// A function to handle overflow and underflow of indices
void Wrapper(int *i, int *j) {
 if ((*i == -1) && (*j == size)) { // If you overflow both
   *i = 0;
   *j = size - 2;
   return;
```

```
 }
 *i = wrap(*i);
 *j = wrap(*j);
}


// A utility used to perform 'intelligent mod'
int wrap(int i) {
 int c = i % size;
 if (c < 0)
   c += size;
 return c;
}


/*************************************************************************
*******/
// A simple utility to print out the matrix
void print_mat(int arr[][size]) {
 for (int i = 0; i < size; i++) {
   for (int j = 0; j < size; j++)
     printf("%d  ", arr[i][j]);
   printf("\n");
 }
 printf("\n");
```

```
}
```

- **Output:**

```
┌──[AnimeshK@kali]─[~/Desktop/GATE_Prep/OS/College/LabAssignments/Exp4]
└─$ ./CreateMagicSquare
Kindly Enter either Odd Size or Doubly Even Size. Singly even order(4*n + 2 type) IS NOT ALLOWED.

Enter the Size N of N X N Magic Square: 17
The Magic Square row by row:
135  117  99  81  63  45  27  9  280  262  244  226  208  190  172  154  153
116  98  80  62  44  26  8  279  261  243  225  207  189  171  170  152  134
97  79  61  43  25  7  278  260  242  224  206  188  187  169  151  133  115
78  60  42  24  6  277  259  241  223  205  204  186  168  150  132  114  96
59  41  23  5  276  258  240  222  221  203  185  167  149  131  113  95  77
40  22  4  275  257  239  238  220  202  184  166  148  130  112  94  76  58
21  3  274  256  255  237  219  201  183  165  147  129  111  93  75  57  39
2  273  272  254  236  218  200  182  164  146  128  110  92  74  56  38  20
289  271  253  235  217  199  181  163  145  127  109  91  73  55  37  19  1
270  252  234  216  198  180  162  144  126  108  90  72  54  36  18  17  288
251  233  215  197  179  161  143  125  107  89  71  53  35  34  16  287  269
232  214  196  178  160  142  124  106  88  70  52  51  33  15  286  268  250
213  195  177  159  141  123  105  87  69  68  50  32  14  285  267  249  231
194  176  158  140  122  104  86  85  67  49  31  13  284  266  248  230  212
175  157  139  121  103  102  84  66  48  30  12  283  265  247  229  211  193
156  138  120  119  101  83  65  47  29  11  282  264  246  228  210  192  174
137  136  118  100  82  64  46  28  10  281  263  245  227  209  191  173  155

┌──[AnimeshK@kali]─[~/Desktop/GATE_Prep/OS/College/LabAssignments/Exp4]
└─$
```

**Thanks.**

Submission by: **CED18I065 (ANIMESH KUMAR)**