Operating Systems - FORK Exercise Assignment-4

1.) Test drive a C program to test drive ORPHAN and ZOMBIE processes

Orphan:

Code:

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

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

   if(pid > 0){
       printf("Parent Process\n");
   }
   else if(pid == 0){
       sleep(10);
       printf("Child Process\n");
   }
   return 0;
}
```

Output:

```
vijay@vijay-desktop:~/Desktop/Operating_Systems-master/forking3rd$ gcc orphan.c -o orphan
vijay@vijay-desktop:~/Desktop/Operating_Systems-master/forking3rd$ ./orphan
Parent Process
vijay@vijay-desktop:~/Desktop/Operating_Systems-master/forking3rd$ Child Process

vijay@vijay-desktop:~/Desktop/Operating_Systems-master/forking3rd$
```

Zombie:

Code:

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

int main(){
   pid_t pid = fork();
   if (pid > 0){
       sleep(10);
       printf("Parent Process\n");
   }
   else if(pid == 0){
       printf("Child Process\n");
       exit(0);
   }
   return 0;
}
```

Output:

```
vijay@vijay-desktop:~/Desktop/Operating_Systems-master/forking3rd$ gcc zombie.c -o zombie
vijay@vijay-desktop:~/Desktop/Operating_Systems-master/forking3rd$ ./zombie
Child Process
Parent Process
vijay@vijay-desktop:~/Desktop/Operating_Systems-master/forking3rd$
```

2.) Develop a multiprocessing version of MERGE or QUICK sort

Code:

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

//Merge Function
```

```
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;
   j = 0;
   k = l;
   while (i < n1 && j < n2){
       if (L[i] <= R[j]){
           arr[k] = L[i];
           i++;
       }
       else{
           arr[k] = R[j];
           j++;
       }
       k++;
   }

   while (i < n1){
       arr[k] = L[i];
       i++;
       k++;
   }

   while (j < n2){
       arr[k] = R[j];
       j++;
       k++;
   }
}
// Sort Function
```

```
void mergeSort(int arr[], int l ,int r){
    if(l < r){
        int m = l + (r-l)/2;
        if(vfork() == 0){
            mergeSort(arr,l,m);
            exit(0);
        }
        else {
            mergeSort(arr,m+1,r);
            merge(arr,l,m,r);
        }
    }
}

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

// Driver Function
int main(){
    int range;
    printf("Enter the size of the input : ");
    scanf("%d",&range);
    int arr[range];
    printf("Enter the input : \n");
    for(int i=0;i<range;i++){
        scanf("%d",&arr[i]);
    }
    printf("%d\n",range);
    printf("Given array is \n");
    printArray(arr, range);
    mergeSort(arr, 0, range - 1);
    printf("\nSorted array is \n");
    printArray(arr, range);
    return 0;
}
```

Output:

```
vijay@vijay-desktop:~/Desktop/Operating_Systems-master/forking3rd$ ./mergeSort
Enter the size of the input : 7
Enter the input :
1
4
8
9
6
3
7
7
Given array is
1 4 8 9 6 3 7

Sorted array is
1 3 4 6 7 8 9
vijay@vijay-desktop:~/Desktop/Operating_Systems-master/forking3rd$
```

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

Code:

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

int main(){
   pid_t pid;
   int count=0;
   while(1){
       pid = vfork();
       if (pid == 0){
           count++;
           exit(0);
       }
```

```
        else if(pid == -1){
            printf("Max Processes allowed :%d\n",count);
            exit(-1);
        }
    }
    return 0;
}
```

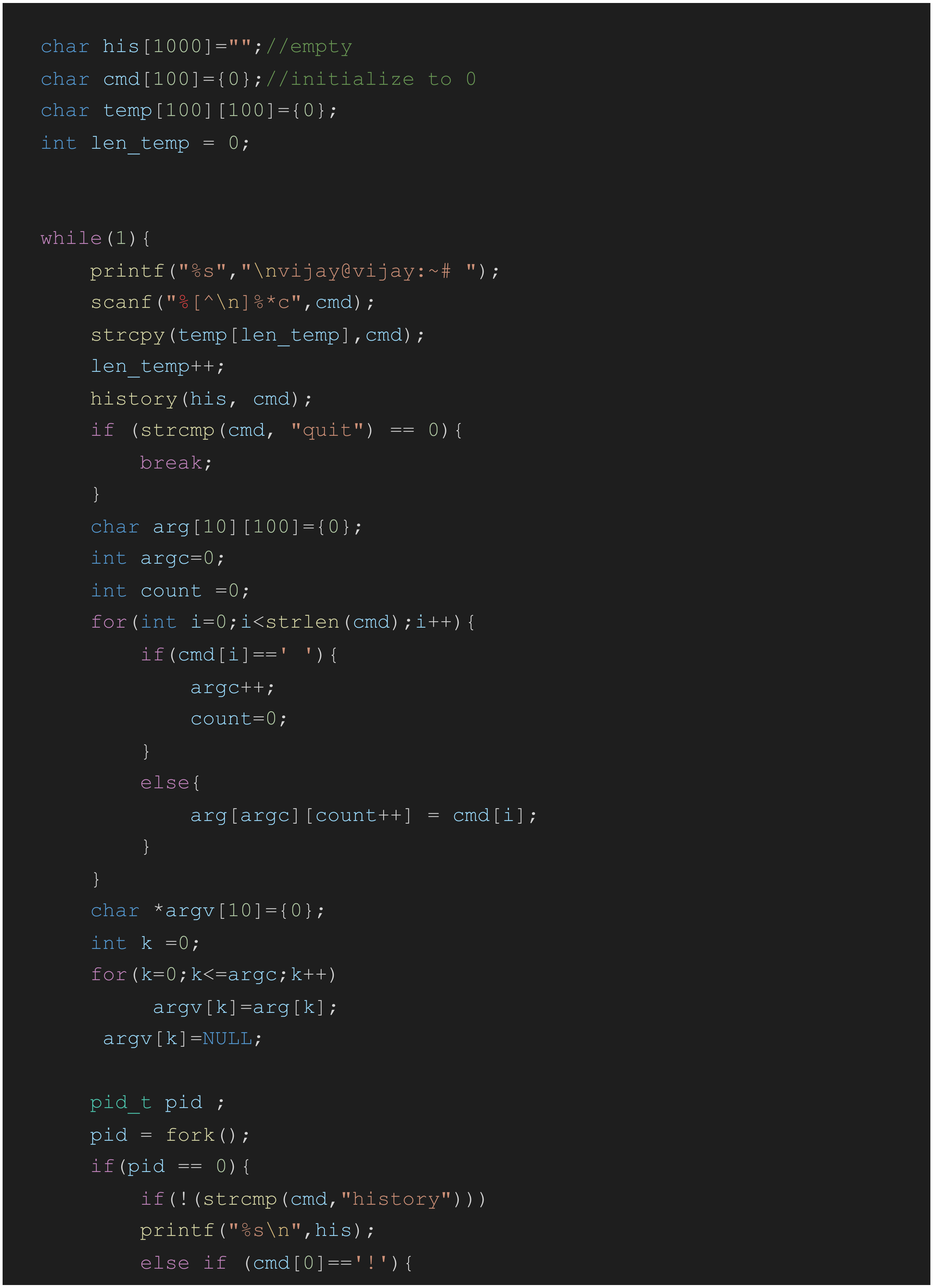
Output:

```
vijay@vijay-desktop:~/Desktop/Operating_Systems-master/forking3rd$ gcc processcount.c -o MaxProcess
vijay@vijay-desktop:~/Desktop/Operating_Systems-master/forking3rd$ ./MaxProcess
Max Processes allowed :20063
vijay@vijay-desktop:~/Desktop/Operating_Systems-master/forking3rd$ ./MaxProcess
Max Processes allowed :20063
vijay@vijay-desktop:~/Desktop/Operating_Systems-master/forking3rd$ ./MaxProcess
Max Processes allowed :20062
vijay@vijay-desktop:~/Desktop/Operating_Systems-master/forking3rd$ ./MaxProcess
Max Processes allowed :20076
vijay@vijay-desktop:~/Desktop/Operating_Systems-master/forking3rd$
```

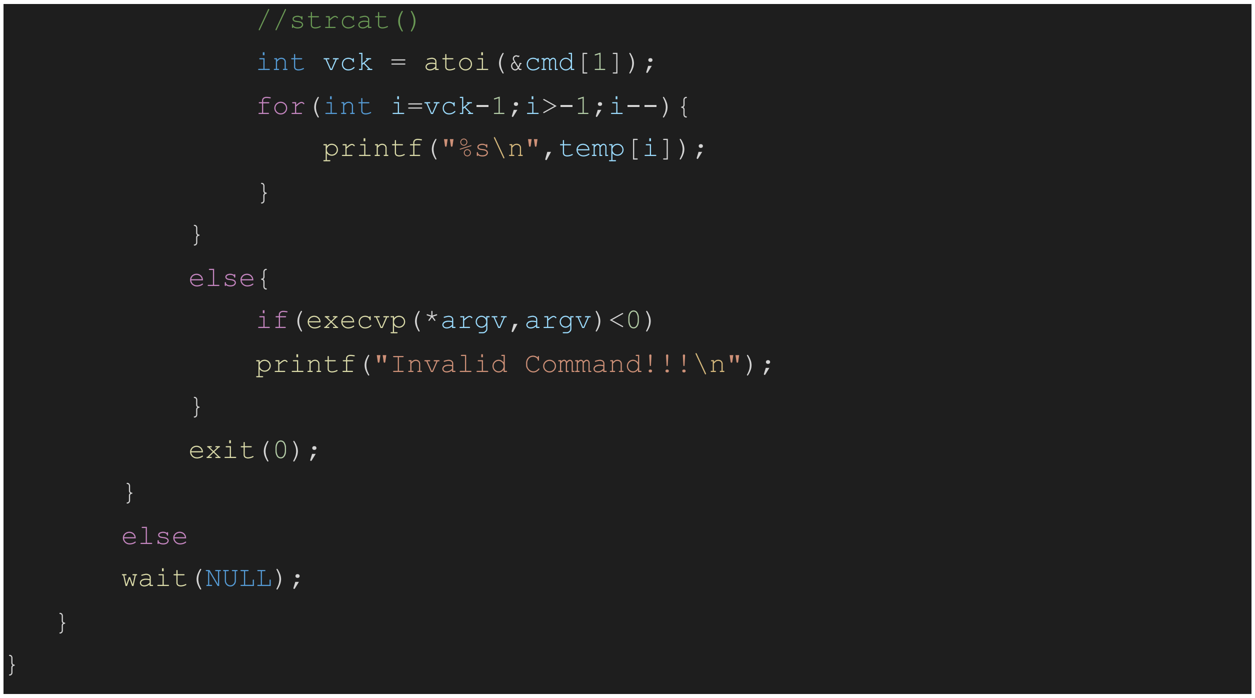
4.) Develop your own command shell [say mark it with @] that accepts user commands(system or user binaries), executes the commands and returns the prompt for further user interaction. Also extend this to support a history feature.

Code:

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

void history(char his[],char cmd[]){
    strcat(his,"\n");
    strcat(his,cmd);
}



int main(){
```



```
char his[1000]="";//empty
char cmd[100]={0};//initialize to 0
char temp[100][100]={0};
int len_temp = 0;


while(1){
    printf("%s","\nvijay@vijay:~# ");
    scanf("%[^\n]%*c",cmd);
    strcpy(temp[len_temp],cmd);
    len_temp++;
    history(his, cmd);
    if (strcmp(cmd, "quit") == 0){
        break;
    }
    char arg[10][100]={0};
    int argc=0;
    int count =0;
    for(int i=0;i<strlen(cmd);i++){
        if(cmd[i]==' '){
            argc++;
            count=0;
        }
        else{
            arg[argc][count++] = cmd[i];
        }
    }
    char *argv[10]={0};
    int k =0;
    for(k=0;k<=argc;k++)
         argv[k]=arg[k];
     argv[k]=NULL;

    pid_t pid ;
    pid = fork();
    if(pid == 0){
        if(!(strcmp(cmd,"history")))
        printf("%s\n",his);
        else if (cmd[0]=='!'){
```



```
                //strcat()
                int vck = atoi(&cmd[1]);
                for(int i=vck-1;i>-1;i--){
                    printf("%s\n",temp[i]);
                }
            }
            else{
                if(execvp(*argv,argv)<0)
                printf("Invalid Command!!!\n");
            }
            exit(0);
        }
        else
        wait(NULL);
    }
}
```

Output:

```
vijay@vijay-desktop:~/Desktop/Operating_Systems-master/forking3rd$ gcc cmd.c -o
cmdnew
vijay@vijay-desktop:~/Desktop/Operating_Systems-master/forking3rd$ ./cmdnew

vijay@vijay:~# ls
CED18I057.pdf  cmdnew   histogram   MaxProcess  mergesort.c  orphan.c       quicksort
zombie
cmd.c          Desktop  magiccheck  mergeSort   orphan       processcount.c  quicksort.c
zombie.c

vijay@vijay:~# mkdir hello

vijay@vijay:~# ls
CED18I057.pdf  cmdnew   hello      magiccheck  mergeSort   orphan    processcount.c
quicksort.c  zombie.c
cmd.c          Desktop  histogram  MaxProcess  mergesort.c  orphan.c  quicksort      zombie

vijay@vijay:~# ls-l
Invalid Command!!!

vijay@vijay:~# ls -l
total 1188
```

```
-rw-rw-r-- 1 vijay vijay 1027729 Nov  1 19:47 CED18I057.pdf
-rw-rw-r-- 1 vijay vijay    1574 Nov 29 14:57 cmd.c
-rwxrwxr-x 1 vijay vijay   17304 Nov 29 15:01 cmdnew
drwxrwxr-x 2 vijay vijay    4096 Nov 29 15:00 Desktop
drwxrwxr-x 2 vijay vijay    4096 Nov 29 15:01 hello
-rw-rw-r-- 1 vijay vijay    8624 Nov  1 19:47 histogram
-rw-rw-r-- 1 vijay vijay   16960 Nov  1 19:47 magiccheck
-rwxrwxr-x 1 vijay vijay   16792 Nov 29 14:51 MaxProcess
-rwxrwxr-x 1 vijay vijay   17072 Nov 29 14:47 mergeSort
-rw-rw-r-- 1 vijay vijay    1670 Nov 29 14:48 mergesort.c
-rwxrwxr-x 1 vijay vijay   16784 Nov 29 14:39 orphan
-rw-rw-r-- 1 vijay vijay     337 Nov 29 14:41 orphan.c
-rw-rw-r-- 1 vijay vijay     438 Nov 29 14:50 processcount.c
-rw-rw-r-- 1 vijay vijay   17040 Nov  1 19:47 quicksort
-rw-rw-r-- 1 vijay vijay    1428 Nov  1 19:47 quicksort.c
-rwxrwxr-x 1 vijay vijay   16824 Nov 29 14:43 zombie
-rw-rw-r-- 1 vijay vijay     354 Nov 29 14:42 zombie.c

vijay@vijay:~# !5
ls -l
ls-l
ls
mkdir hello
ls

vijay@vijay:~# history

ls
mkdir hello
ls
ls-l
ls -l
!5
history

vijay@vijay:~# quit
vijay@vijay-desktop:~/Desktop/Operating_Systems-master/forking3rd$
```

5.) Develop a multiprocessing version of histogram generator to count occurrences of various characters in a given text.

Code:

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

int main(){
   char input[512];
   while(1){
       printf("Enter the input : ");
       scanf("%s",input);
       int count[128] ;
       for(int i =0; i<128;i++){
           count[i]=0;
       }

       pid_t pid;
       pid = vfork();

       if(pid == 0){
           for(int i = 0;i<strlen(input);i++){
               count[(int)input[i]]++;
           }
           exit(0);
       }
       else if (pid > 0){
           wait(NULL);
           for(int i = 0;i<128;i++){
               printf("%c => %d",(char)i,count[i]);
               for(int j=0;j<count[i];j++)printf("#");
               printf("\n");
           }
           printf("\n");
           int flag;
           printf("Do you want to exit: 1:exit | 0:repeat ");
           scanf("%d",&flag);
           if(flag == 1){
               return 0;
```

```
            }
        }
    }
    return 0;
}
```

Output:

```
vijay@vijay-desktop:~/Desktop/Operating_Systems-master/forking3rd/Desktop$ ./hist
Enter the input : qwertyuiop1234567890!@#$%^&*()\|/;'":
 => 0
 => 0
 => 0
 => 0
 => 0
 => 0
 => 0
 => 0
 => 0
        => 0

 => 0

 => 0

 => 0
 => 0
 => 0
 => 0
 => 0
 => 0
 => 0
 => 0
 => 0
 => 0
 => 0
 => 0
 => 0
 => 0
 => 0
> 0
 => 0
 => 0
 => 0
 => 0
```

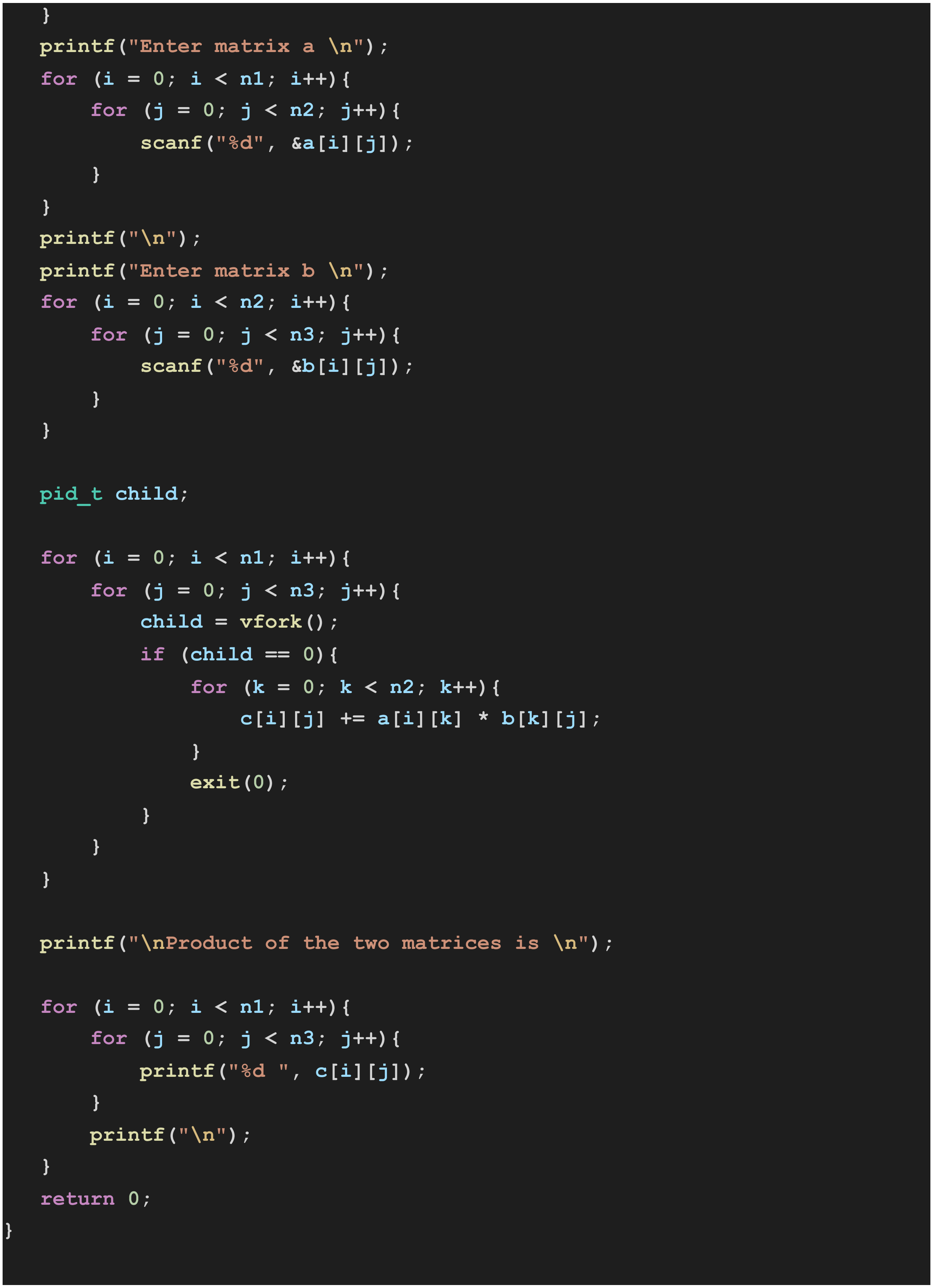
```
  => 0
! => 1#
" => 1#
# => 1#
$ => 1#
% => 1#
& => 1#
' => 1#
( => 1#
) => 1#
* => 1#
+ => 0
, => 0
- => 0
. => 0
/ => 1#
0 => 1#
1 => 1#
2 => 1#
3 => 1#
4 => 1#
5 => 1#
6 => 1#
7 => 1#
8 => 1#
9 => 1#
: => 1#
; => 1#
< => 0
= => 0
> => 0
? => 0
@ => 1#
A => 0
B => 0
C => 0
D => 0
E => 0
F => 0
G => 0
H => 0
I => 0
J => 0
K => 0
```

```
L => 0
M => 0
N => 0
O => 0
P => 0
Q => 0
R => 0
S => 0
T => 0
U => 0
V => 0
W => 0
X => 0
Y => 0
Z => 0
[ => 0
\ => 1#
] => 0
^ => 1#
_ => 0
` => 0
a => 0
b => 0
c => 0
d => 0
e => 1#
f => 0
g => 0
h => 0
i => 1#
j => 0
k => 0
l => 0
m => 0
n => 0
o => 1#
p => 1#
q => 1#
r => 1#
s => 0
t => 1#
u => 1#
v => 0
w => 1#
```

```
x => 0
y => 1#
z => 0
{ => 0
| => 1#
} => 0
~ => 0
 => 0

Do you want to exit: 1:exit | 0:repeat 1
vijay@vijay-desktop:~/Desktop/Operating_Systems-master/forking3rd/Desktop$
```

6.) Develop a multiprocessing version of the matrix multiplication
Code:

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

int main(){
   int n1, n2, n3;
   printf("Enter no. of rows of first matrix: ");
   scanf("%d", &n1);

   printf("Enter no. of rows of second matrix: ");
   scanf("%d", &n2);

   printf("\n Enter no. of columns of second matrix: ");
   scanf("%d", &n3);

   int i, j, k;
   int a[n1][n2], b[n2][n3], c[n1][n3];

   for (i = 0; i < n1; i++){
       for (j = 0; j < n3; j++){
           c[i][j] = 0;
       }
```



```
    }
    printf("Enter matrix a \n");
    for (i = 0; i < n1; i++){
        for (j = 0; j < n2; j++){
            scanf("%d", &a[i][j]);
        }
    }
    printf("\n");
    printf("Enter matrix b \n");
    for (i = 0; i < n2; i++){
        for (j = 0; j < n3; j++){
            scanf("%d", &b[i][j]);
        }
    }

    pid_t child;

    for (i = 0; i < n1; i++){
        for (j = 0; j < n3; j++){
            child = vfork();
            if (child == 0){
                for (k = 0; k < n2; k++){
                    c[i][j] += a[i][k] * b[k][j];
                }
                exit(0);
            }
        }
    }

    printf("\nProduct of the two matrices is \n");

    for (i = 0; i < n1; i++){
        for (j = 0; j < n3; j++){
            printf("%d ", c[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

Output:

```
vijay@vijay-desktop:~/Desktop/Operating_Systems-master/forking3rd/Desktop$ ./matrixMult
Enter no. of rows of first matrix: 3
Enter no. of rows of second matrix: 3

 Enter no. of columns of second matrix: 3
Enter matrix a
1
2
3
4
5
6
7
8
9

Enter matrix b
9
8
7
6
5
4
3
2
1

Product of the two matrices is
30 24 18
84 69 54
138 114 90
vijay@vijay-desktop:~/Desktop/Operating_Systems-master/forking3rd/Desktop$
```

7.) Develop a parallelized application to check for if a user input square matrix is a magic square or not , extend the code to support magic square generation (input will be order of matrix).(Extra Credit).

Magic Square checker

Code:

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

```
#include <math.h>
#include <string.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

int isMagicSquare(int *mat, int N){
    // calculate the sum of the prime diagonal
    int sum = 0, sum2 = 0;

    int diag_pid = vfork();
    if (diag_pid == 0){
        for (int i = 0; i < N; i++)
            sum = sum + *((mat + i * N) + i);
        exit(0);
    }
    else if (diag_pid > 0){
        wait(NULL);
        // the secondary diagonal
        for (int i = 0; i < N; i++)
            sum2 = sum2 + *((mat + i * N) + (N - 1 - i));
        if (sum != sum2)
            return 0;
    }

    int row_pid = vfork();
    if (row_pid == 0){
        // For sums of Rows
        for (int i = 0; i < N; i++){

            int rowSum = 0;
            for (int j = 0; j < N; j++)
                rowSum += *((mat + i * N) + j);

            // check if every row sum is equal to prime diagonal sum
            if (rowSum != sum)
                return 0;
        }
        exit(0);
    }
    else if (row_pid > 0){
        wait(NULL);
        // For sums of Columns
        for (int i = 0; i < N; i++){
            int colSum = 0;
            for (int j = 0; j < N; j++)
                colSum += *((mat + j * N) + i);
            // check if every column sum is equal to prime diagonal sum
            if (sum != colSum)
                return 0;
        }
    }

    return 1;
}

int main(){
    int n, i, j;

    printf("Enter order of matrix:-\n");
    scanf("%d", &n);

    int A[n][n];
    printf("Enter matrix:-\n");

    for (i = 0; i < n; i++){
        for (j = 0; j < n; j++){
            scanf("%d", &A[i][j]);
        }
    }
    if (isMagicSquare((int *)A, n)){
        printf("Magic Square\n");
    }
    else if(isMagicSquare){
        printf("Not a Magic Sqaure\n");
    }
    return 0;
}
```

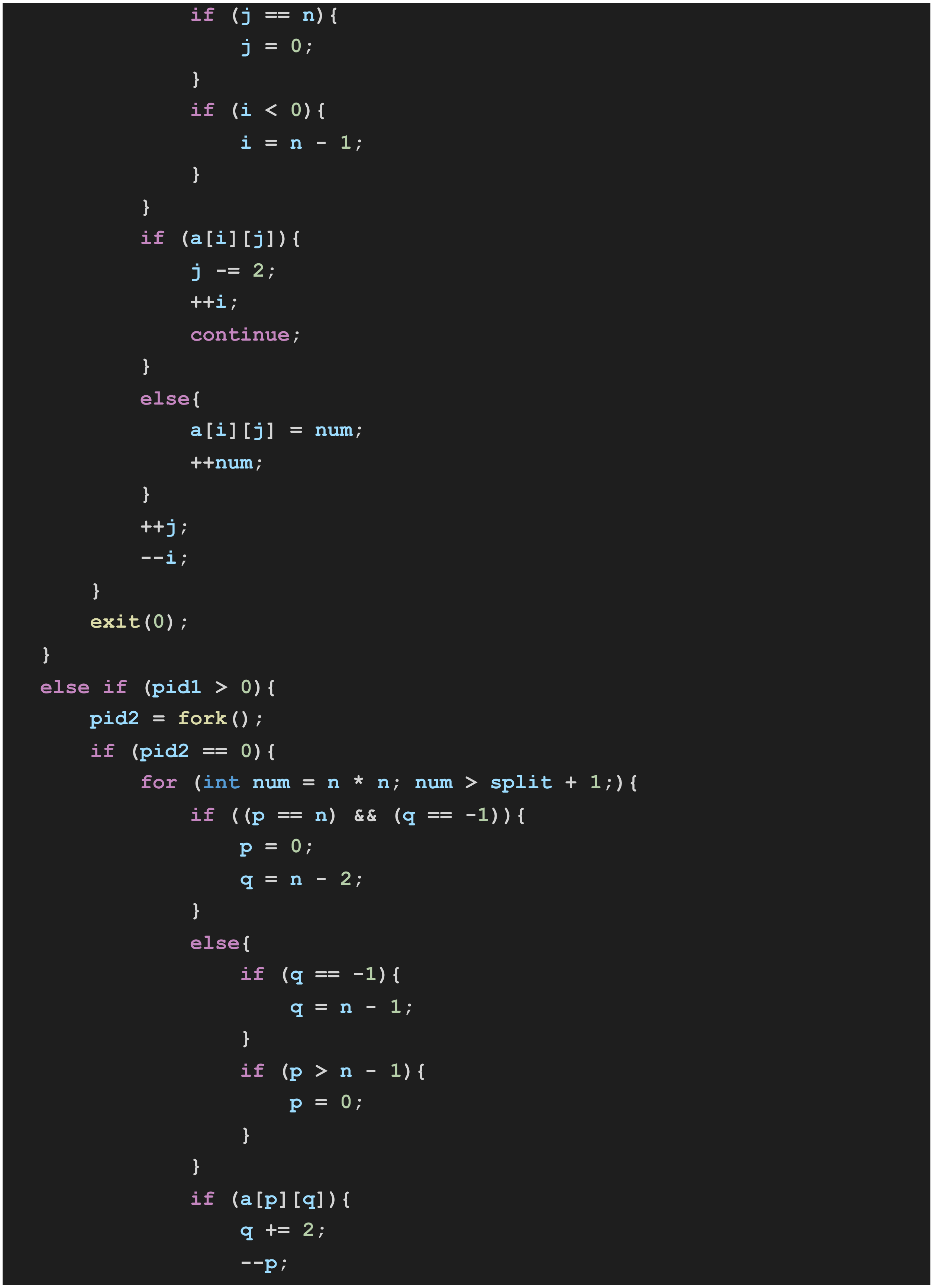
Output:

```
vijay@vijay-desktop:~/Desktop/Operating_Systems-master/forking3rd/Desktop$ ./magicSquare
Enter order of matrix:-
3
Enter matrix:-
1 2 3 4 5 6 7 8 9
Not a Magic Sqaure
Segmentation fault (core dumped)
vijay@vijay-desktop:~/Desktop/Operating_Systems-master/forking3rd/Desktop$ ./magicSquare
Enter order of matrix:-
3
Enter matrix:-
2 7 6 9 5 1 4 3 8
Magic Square
vijay@vijay-desktop:~/Desktop/Operating_Systems-master/forking3rd/Desktop$
```

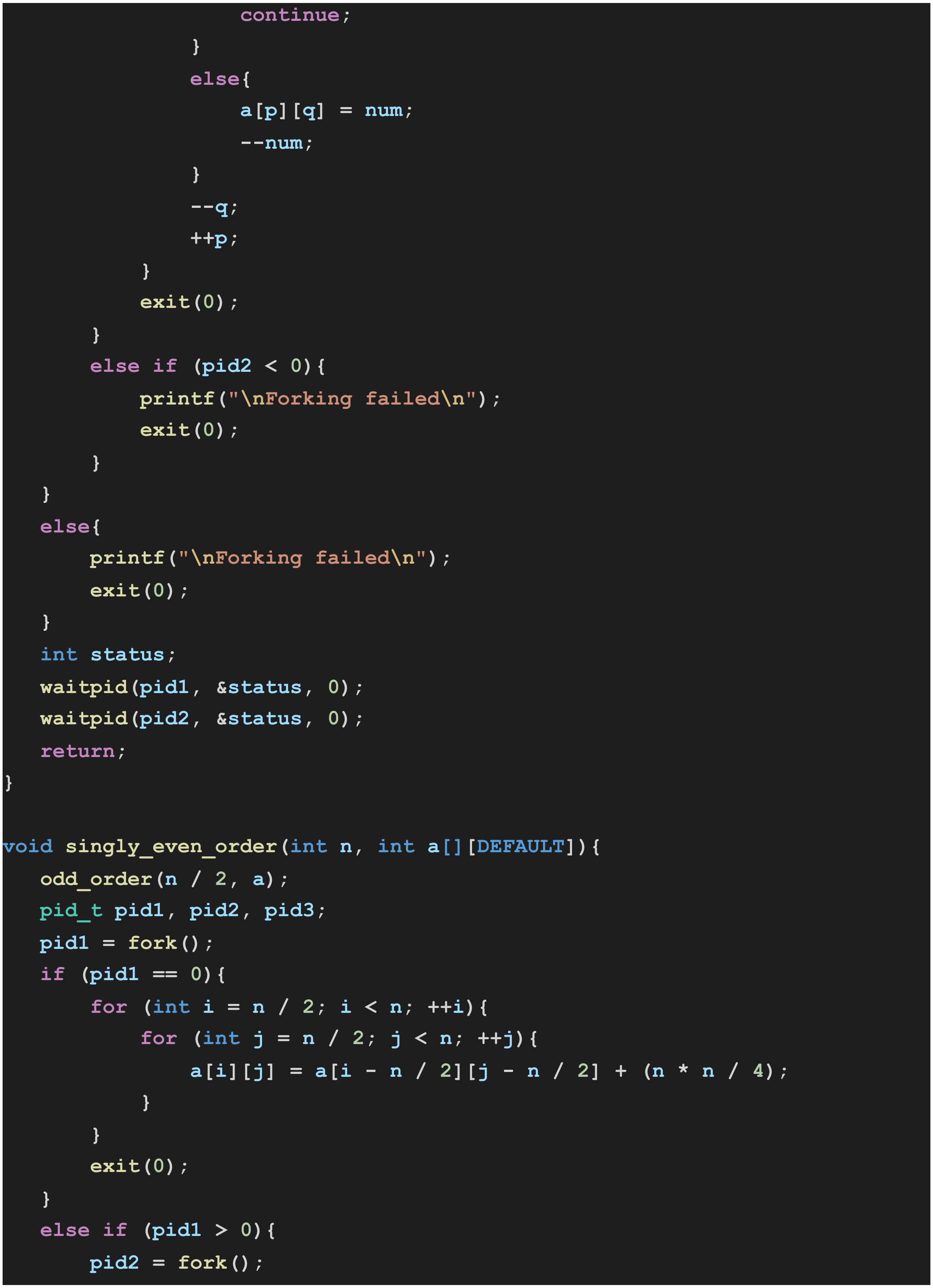
Magic Square Generator

Code:

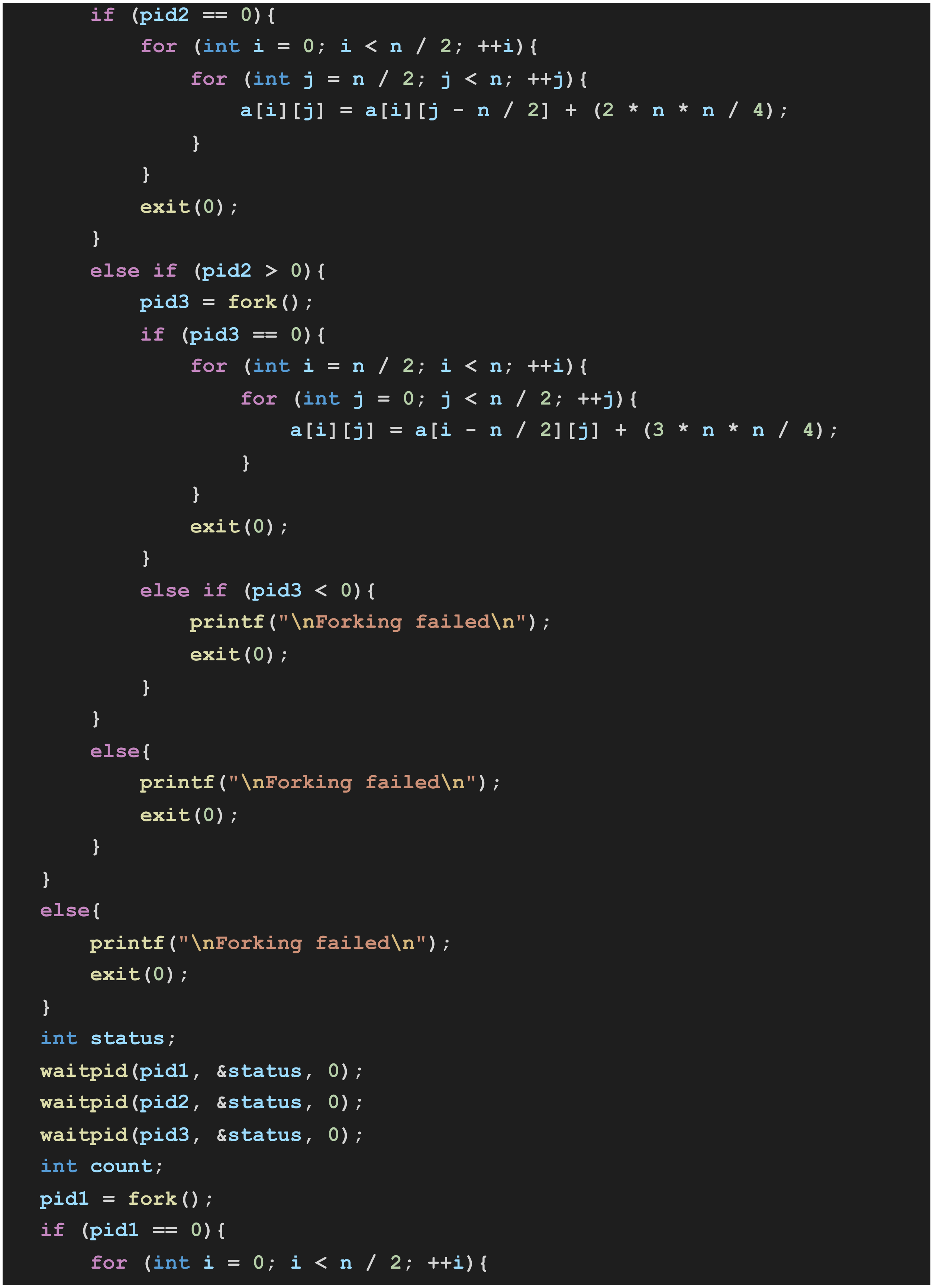
```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <sys/shm.h>
#include <unistd.h>

#define DEFAULT 100

void odd_order(int n, int a[][DEFAULT]){
   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{
```



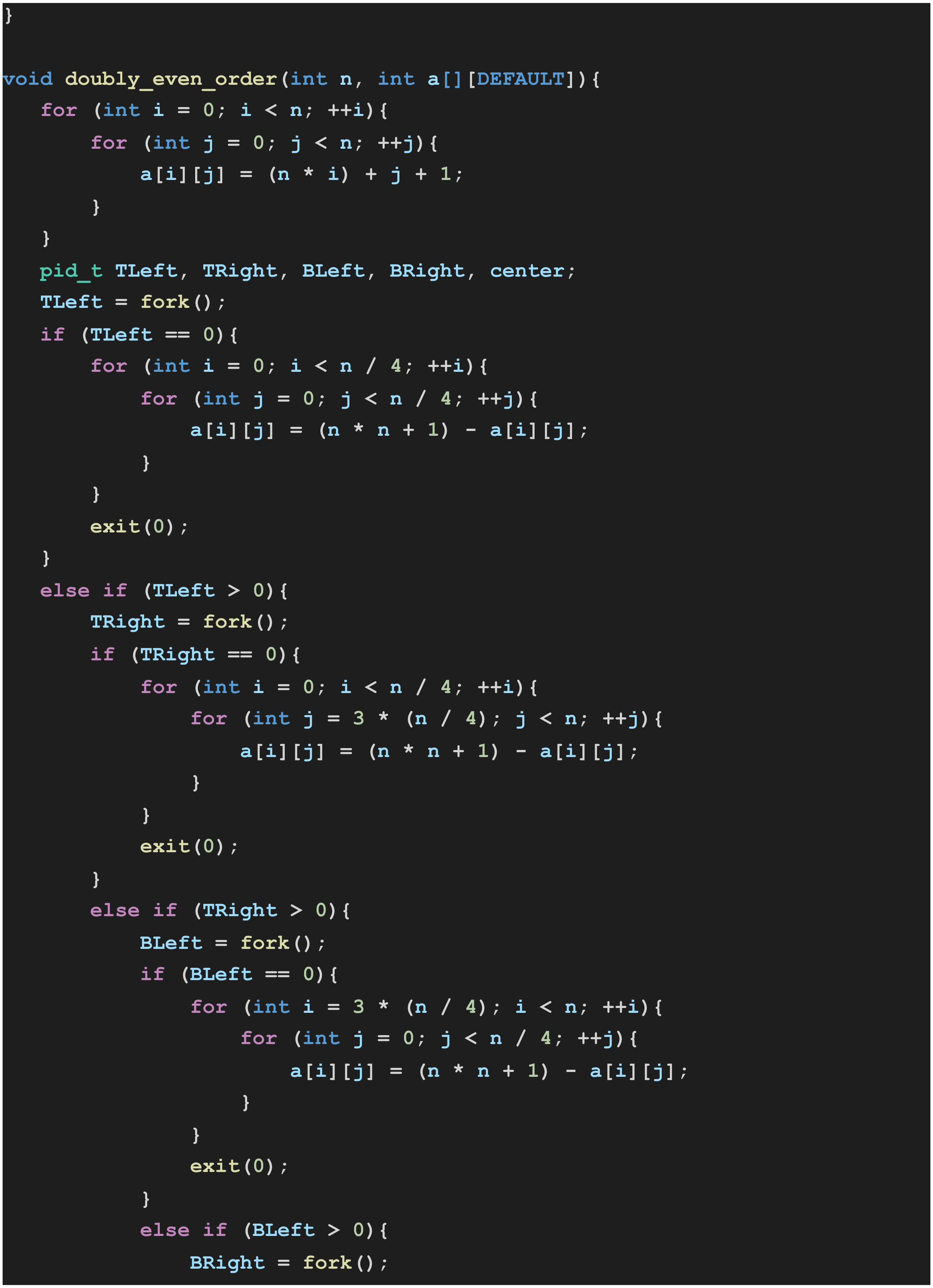
```
            if (j == n){
                j = 0;
            }
            if (i < 0){
                i = n - 1;
            }
        }
        if (a[i][j]){
            j -= 2;
            ++i;
            continue;
        }
        else{
            a[i][j] = num;
            ++num;
        }
        ++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 == -1){
                    q = n - 1;
                }
                if (p > n - 1){
                    p = 0;
                }
            }
            if (a[p][q]){
                q += 2;
                --p;
```



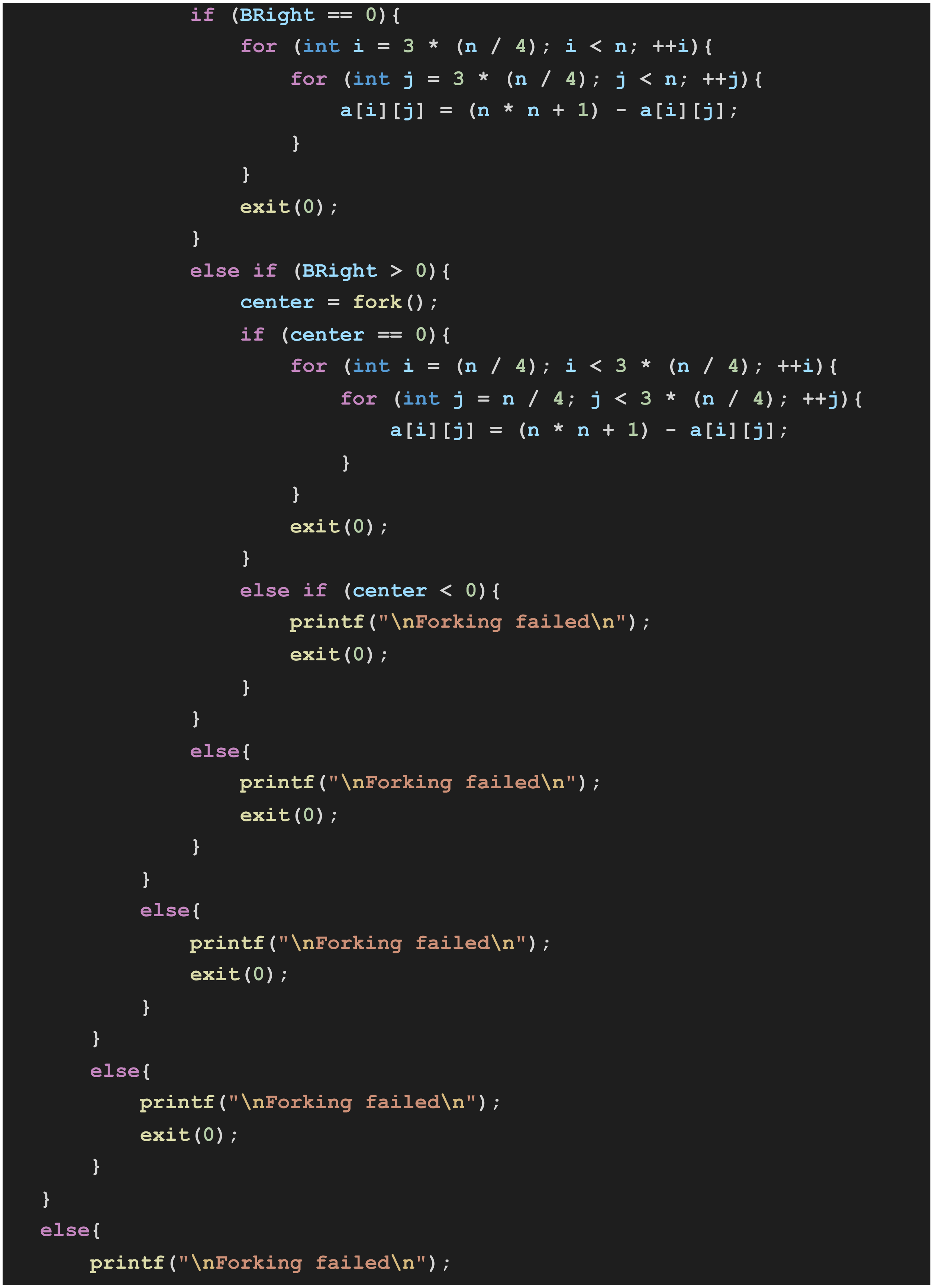
```
                    continue;
                }
                else{
                    a[p][q] = num;
                    --num;
                }
                --q;
                ++p;
            }
            exit(0);
        }
        else if (pid2 < 0){
            printf("\nForking failed\n");
            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[][DEFAULT]){
    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;
```

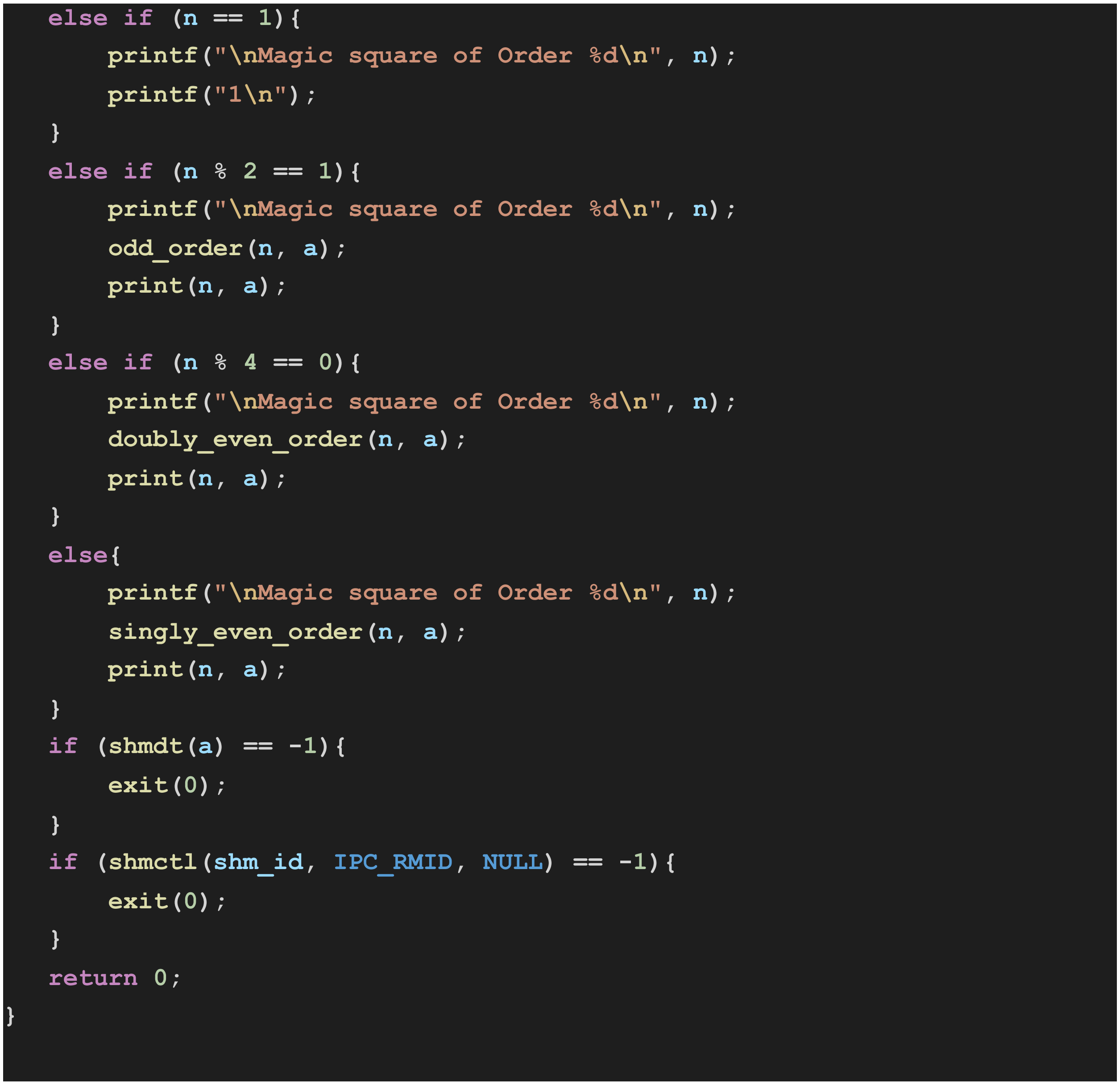


```
}

void doubly_even_order(int n, int a[][DEFAULT]){
    for (int i = 0; i < n; ++i){
        for (int j = 0; j < n; ++j){
            a[i][j] = (n * i) + j + 1;
        }
    }
    pid_t TLeft, TRight, BLeft, BRight, center;
    TLeft = fork();
    if (TLeft == 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 (TLeft > 0){
        TRight = fork();
        if (TRight == 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 (TRight > 0){
            BLeft = fork();
            if (BLeft == 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 (BLeft > 0){
                BRight = fork();
```



```
            if (BRight == 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 (BRight > 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);
    }
}
else{
    printf("\nForking failed\n");
```

```
        exit(0);
    }
    int status;
    waitpid(TLeft, &status, 0);
    waitpid(TRight, &status, 0);
    waitpid(BLeft, &status, 0);
    waitpid(BRight, &status, 0);
    waitpid(center, &status, 0);
    return;
}

void print(int n, int a[][DEFAULT]){
    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("\nSHM Failed\n");
        exit(0);
    }
    a = shmat(shm_id, 0, 0);
    if (a == (void *)-1){
        printf("\nSHM 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("1\n");
    }
    else if (n % 2 == 1){
        printf("\nMagic square of Order %d\n", n);
        odd_order(n, a);
        print(n, a);
    }
    else if (n % 4 == 0){
        printf("\nMagic square of Order %d\n", n);
        doubly_even_order(n, a);
        print(n, a);
    }
    else{
        printf("\nMagic square of Order %d\n", n);
        singly_even_order(n, a);
        print(n, a);
    }
    if (shmdt(a) == -1){
        exit(0);
    }
    if (shmctl(shm_id, IPC_RMID, NULL) == -1){
        exit(0);
    }
    return 0;
}
```

Output:

```
vijay@vijay-desktop:~/Desktop/Operating_Systems-master/forking3rd/Desktop$ gcc
magicgenerate.c -o magicGen
vijay@vijay-desktop:~/Desktop/Operating_Systems-master/forking3rd/Desktop$ ./magicGen

Enter the order of Magic square : 3

Magic square of Order 3
2 7 6
9 5 1
4 3 8
```

```
vijay@vijay-desktop:~/Desktop/Operating_Systems-master/forking3rd/Desktop$ ./magicGen

Enter the order of Magic square : 4

Magic square of Order 4
16  2   3   13
5   11  10  8
9    7    6  12
4   14  15   1
vijay@vijay-desktop:~/Desktop/Operating_Systems-master/forking3rd/Desktop$
```