

OS Assignment 5

Class Codes:

Code1 (Pipe code 1):

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

#define BUFFER_SIZE 25
#define READ_END 0
#define WRITE_END 1

int main(int argc, char const *argv[])
{
    char write_msg [BUFFER_SIZE] = "Greetings";
    char read_msg [BUFFER_SIZE];
    int fd[2];
    pid_t pid;

    /* create the pipe */
    if (pipe (fd) == -1)
    {
        fprintf(stderr, "Pipe failed");
        return 1;
    }

    pid = fork();

    if (pid < 0)
    {
        /* error occurred */
        fprintf(stderr, "Fork Failed");
        return 1;
    }
```

```

if (pid > 0)
{
    /* parent process */
    /* close the unused end of the pipe */
    close(fd[READ_END]);

    /* write to the pipe */
    write(fd[WRITE_END], write_msg, strlen(write_msg) + 1);

    /* close the write end of the pipe */
    close(fd [WRITE_END]);
}

else
{
    /* child process */
    /* close the unused end of the pipe */
    close(fd [WRITE_END]);

    /* read from the pipe */
    read(fd [READ_END], read_msg, BUFFER_SIZE);

    printf("read %s\n",read_msg);

    /* close the write end of the pipe */
    close(fd [READ_END]);
}

return 0;

}

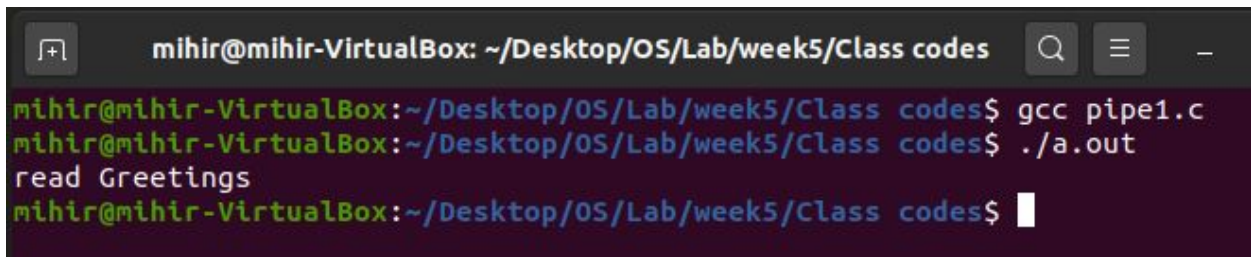
```

Explanation:

We create a pipe and 2 processes, parent and child. Both the processes have their own read and write ends of the pipe. Since the parent has to write the data, the read end of the pipe in the parent process is closed. Similarly, since the child has to read the data, the write end of the pipe in the parent process is closed.

Parent writes the message to the write end of the pipe which is read by the child through the pipe's read end.

Output ss:



```
mihir@mihir-VirtualBox: ~/Desktop/OS/Lab/week5/Class codes
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week5/Class codes$ gcc pipe1.c
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week5/Class codes$ ./a.out
read Greetings
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week5/Class codes$
```

Code 2 (Pipe code 2):

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

int main(int argc, char const *argv[])
{
    int pipefds1[2], pipefds2[2];
    int returnstatus1, returnstatus2;
    int pid;
    char pipe1writemessage[20] = "Hi";
    char pipe2writemessage[20] = "Hello";
    char readmessage[20];

    returnstatus1 = pipe(pipefds1);

    if (returnstatus1 == -1)
    {
        printf("Unable to create Pipe 1\n");
        return 1;
    }

    returnstatus2 = pipe(pipefds2);

    if (returnstatus2 == -1)
    {
        printf("Unable to create Pipe 2\n");
```

```

        return 1;
    }

    pid = fork();

    if (pid > 0)
    {
        close(pipefds1[0]);
        close(pipefds2[1]);

        printf("Parent writing to pipe 1. Message sent to child is - %s\n",
pipe1writemessage);

        write(pipefds1[1], pipe1writemessage, sizeof(pipe1writemessage)+1);

        read(pipefds2[0], readmessage, sizeof(readmessage));

        printf("Parent read from pipe 2. Message read from child is - %s\n",
readmessage);
    }

    else
    {
        close(pipefds1[1]);
        close(pipefds2[0]);

        read(pipefds1[0], readmessage, sizeof(readmessage));

        printf("Child read from pipe 2. Message read from parent is - %s\n",
readmessage);

        printf("Child writing to pipe 1. Message sent to parent is - %s\n",
pipe2writemessage);

        write(pipefds2[1], pipe2writemessage, sizeof(pipe2writemessage)+1);

    }

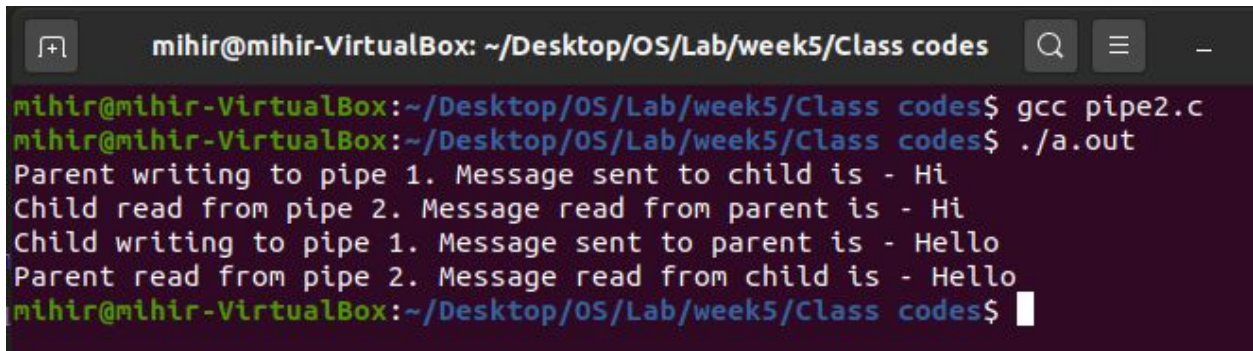
    return 0;
}

```

Explanation:

2 pipes are used. One for the parent to write and child to read while another for the parent to read and child to write. Parent writes a message to the write end of pipe 1 which is read by the child from the read end of pipe 1. The child writes a message to the write end of pipe 2 which is read by the parent from the read end of pipe 2. The unused ends of both the pipes in both the processes are closed.

Output ss:

A terminal window titled 'mihir@mihir-VirtualBox: ~/Desktop/OS/Lab/week5/Class codes'. The prompt is 'mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week5/Class codes\$'. The user enters 'gcc pipe2.c' and then './a.out'. The program output is: 'Parent writing to pipe 1. Message sent to child is - Hi', 'Child read from pipe 2. Message read from parent is - Hi', 'Child writing to pipe 1. Message sent to parent is - Hello', 'Parent read from pipe 2. Message read from child is - Hello'. The prompt returns at the end.

```
mihir@mihir-VirtualBox: ~/Desktop/OS/Lab/week5/Class codes
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week5/Class codes$ gcc pipe2.c
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week5/Class codes$ ./a.out
Parent writing to pipe 1. Message sent to child is - Hi
Child read from pipe 2. Message read from parent is - Hi
Child writing to pipe 1. Message sent to parent is - Hello
Parent read from pipe 2. Message read from child is - Hello
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week5/Class codes$
```

Code 3 (dup2 code 1):

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

int main(void)
{
    int number1, number2, sum;
    int input_fds = open("./input.txt", O_RDONLY);

    if (dup2(input_fds, STDIN_FILENO) < 0)
    {
        printf("Unable to duplicate file descriptor.");
        exit(EXIT_FAILURE);
    }
}
```

```

scanf("%d %d", &number1, &number2);
sum = number1 + number2;

printf("%d + %d = %d\n", number1, number2, sum);

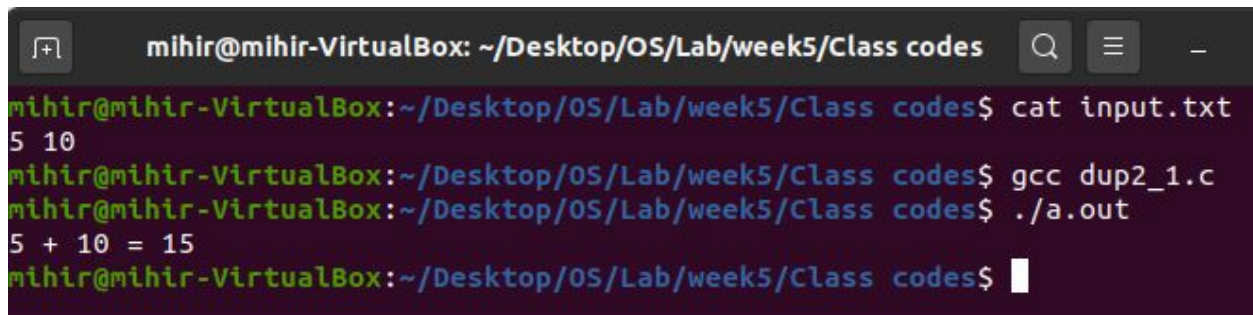
return EXIT_SUCCESS;
}

```

Explanation:

The 2 input numbers are read from a file called “input.txt” instead of supplying them through STDIN. Possible because of dup2.

Output ss:



```

mihir@mihir-VirtualBox: ~/Desktop/OS/Lab/week5/Class codes
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week5/Class codes$ cat input.txt
5 10
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week5/Class codes$ gcc dup2_1.c
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week5/Class codes$ ./a.out
5 + 10 = 15
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week5/Class codes$

```

Code 4 (dup2 code 2):

```

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int main(void)
{
    int number1, number2, sum;
    int input_fds = open("./input.txt", O_RDONLY);

```

```

int output_fds = open("./output.txt", O_WRONLY);

dup2(input_fds, STDIN_FILENO); //0
dup2(output_fds, STDOUT_FILENO); //1

scanf("%d %d", &number1, &number2);
sum = number1 + number2;

printf("%d + %d = %d\n", number1, number2, sum);

return EXIT_SUCCESS;
}

```

Explanation:

2 calls to dup2 are made. One for taking input through a file “input.txt” instead of through STDIN and another for supplying the output to a file “output.txt” instead of supplying it to STDOUT.

Output ss:

```

mihir@mihir-VirtualBox: ~/Desktop/OS/Lab/week5/Class codes
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week5/Class codes$ ls
a.out  dup2_1.c  dup2_2.c  input.txt  output.txt  pipe1.c  pipe2.c
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week5/Class codes$ cat input.txt
5 10
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week5/Class codes$ gcc dup2_2.c
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week5/Class codes$ ./a.out
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week5/Class codes$ cat output.txt
5 + 10 = 15
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week5/Class codes$

```

Code 5 (dup2 code 3):

```

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

```

```

#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int main()
{
    // open() returns a file descriptor file_desc to a // the file "dup.txt" here
    int file_desc = open("dup.txt", O_WRONLY | O_APPEND);

    if(file_desc < 0) printf("Error opening the file\n");

    int copy_desc = dup(file_desc);

    write(copy_desc, "This will be output to the file named dup.txt\n", 46);
    write(file_desc, "This will also be output to the file named dup.txt\n", 51);

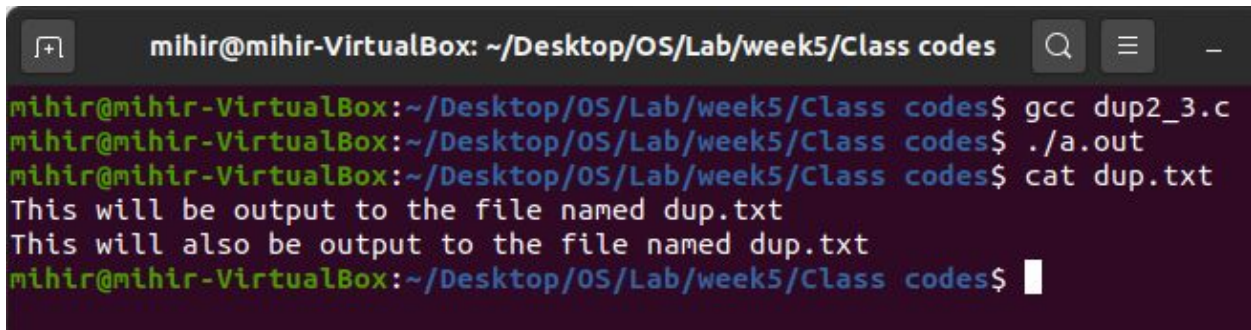
    return 0;
}

```

Explanation:

File descriptor is copied through a call made to dup() into another file descriptor but both are pointing to the same file.

Output ss:



```

mihir@mihir-VirtualBox: ~/Desktop/OS/Lab/week5/Class codes
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week5/Class codes$ gcc dup2_3.c
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week5/Class codes$ ./a.out
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week5/Class codes$ cat dup.txt
This will be output to the file named dup.txt
This will also be output to the file named dup.txt
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week5/Class codes$

```


Code 6 (dup2 code 4):

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

#define MAX 512

int main(int argc, char* argv[])
{
    int fd[2];
    char buf[MAX];
    int nb, i;

    if (pipe(fd) == -1)
    {
        perror("Creating pipe");
        exit(1);
    }

    switch(fork())
    {
        case -1:
            perror("Creating a process ");
            exit(1);

        case 0:
            dup2(fd[1], 1);
            execvp("ls", argv); // output of ls command written on fd[1] courtesy the
dup2 call.
            perror("program ls");
            exit(1);

        default:
            close(fd[1]);

            while ((nb=read(fd[0], buf, MAX)) > 0)
            {
                for(i=0; i<nb; i++)
```

```

        buf[i] = toupper(buf[i]);

    if (write(1, buf, nb) == -1)
    {
        perror ("Writing to stdout");
        exit(1);
    }

} // end of while

if (nb == -1)
{
    perror("Reading from pipe");
    exit(1);
}

} // end of switch

} // main end

```

Explanation:

This is implemented using dup2 and pipe. The output of ls is sent to the write end of the pipe instead of sending it to the STDOUT through dup2 in the child process.

In the parent process, this output is read character by character and converted to upper case and then sent to STDOUT.

Output ss:

```

mihir@mihir-VirtualBox: ~/Desktop/OS/Lab/week5/Class codes
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week5/Class codes$ gcc dup2_4.c
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week5/Class codes$ ./a.out
A.OUT
DUP2_1.C
DUP2_2.C
DUP2_3.C
DUP2_4.C
DUP.TXT
INPUT.TXT
OUTPUT.TXT
PIPE1.C
PIPE2.C
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week5/Class codes$

```

Code 7 (dup2 code 5):

```
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
#include<ctype.h>
int main ()
{
    char *cmd1[] = { "/bin/ls", 0 };
    char *cmd2[] = { "/usr/bin/tr", "a-z", "A-Z", 0 };
    int pid; int pfd[2];

    if (pipe(pfd) == -1)
    {
        perror("Creating pipe");
        exit(1);
    }

    switch (pid = fork())
    {
        case 0: /* child */
            dup2(pfd[0], 0);
            close(pfd[1]); /* the child does not need this end of the pipe */
            execvp(cmd2[0], cmd2); // executes tr command on output of ls stored at
the respective pipe end
            perror(cmd2[0]);

            default: /* parent */
                dup2(pfd[1], 1);
                close(pfd[0]);
                /* the parent does not need this end of the pipe */
                execvp(cmd1[0], cmd1);
                perror(cmd1[0]);
                // ls executed and written to the pipe end of 1

        case -1:
            perror("fork");
            exit(1);
    }
}
```

Explanation:

This is an alternative approach to the previous program. The logic is the same, just that the case conversion is not done with a loop but by calling tr command through execvp.

Output ss:

```
mihir@mihir-VirtualBox: ~/Desktop/OS/Lab/week5/Class codes
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week5/Class codes$ gcc dup2_5.c
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week5/Class codes$ ./a.out
A.OUT
DUP2_1.C
DUP2_2.C
DUP2_3.C
DUP2_4.C
DUP2_5.C
DUP.TXT
INPUT.TXT
OUTPUT.TXT
PIPE1.C
PIPE2.C
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week5/Class codes$
```

Assignment Codes:

(1) Parent sets up a string which is read by the child, reversed there and read back to the parent.

Logic: 2 pipes are used. One for the parent to write and child to read while another for the parent to read and child to write. Parent writes a message to the write end of pipe 1 which is read by the child from the read end of pipe 1. The child reverses the string and sends it back to the parent by writing it to the write end of pipe 2 which is read by the parent from the read end of pipe 2.

The unused ends of both the pipes in both the processes are closed.

Code:

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

#define R_END 0
#define W_END 1

int main(int argc, char const *argv[])
{
    int fd1[2], fd2[2];
    int returnstatus1, returnstatus2;
    int pid;
    char pipe1msg[9] = "MihirShri";
    char pipe2msg[9];
    char readmsg[9];

    returnstatus1 = pipe(fd1);

    if (returnstatus1 == -1)
    {
        printf("Unable to create Pipe 1\n");
        return 1;
    }

    returnstatus2 = pipe(fd2);

    if (returnstatus2 == -1)
    {
        printf("Unable to create Pipe 2\n");
        return 1;
    }

    pid = fork();

    if (pid > 0)
    {
        close(fd1[R_END]);
        close(fd2[W_END]);

        printf("Parent writing to pipe 1. Message sent to child is - %s\n\n", pipe1msg);
```

```

        write(fd1[W_END], pipe1msg, sizeof(pipe1msg)+1);

        read(fd2[R_END], readmsg, sizeof(readmsg));

        printf("Parent read from pipe 2. Message read from child is - %s\n", readmsg);
    }

    else
    {
        close(fd1[W_END]);
        close(fd2[R_END]);

        read(fd1[R_END], readmsg, sizeof(readmsg));

        int j=0;
        int i=sizeof(readmsg)/sizeof(readmsg[0]) -1;

        for(;i>=0;i--)
        {
            pipe2msg[j] = readmsg[i];
            j++;
        }

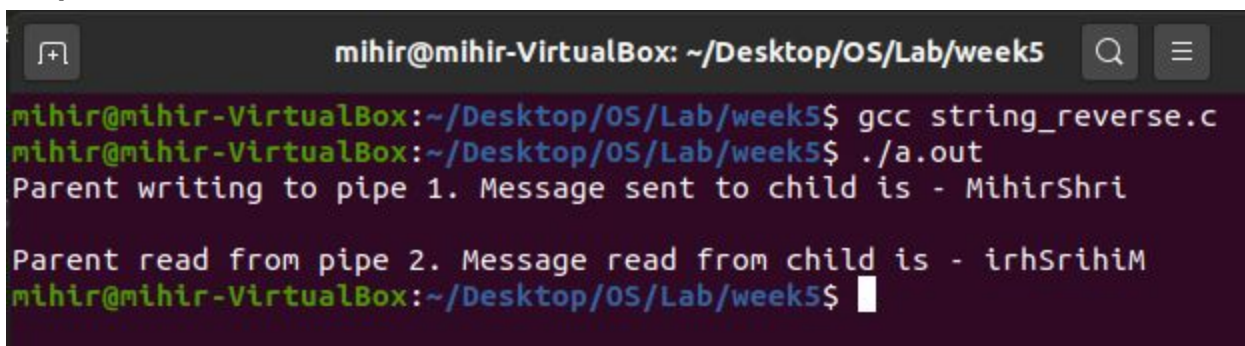
        write(fd2[W_END], pipe2msg, sizeof(pipe2msg)+1);

    }

    return 0;
}

```

Output ss:



```

mihir@mihir-VirtualBox: ~/Desktop/OS/Lab/week5
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week5$ gcc string_reverse.c
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week5$ ./a.out
Parent writing to pipe 1. Message sent to child is - MihirShri

Parent read from pipe 2. Message read from child is - irhSrihiM
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week5$

```

(2) Parent sets up string 1 and child sets up string 2. String 2 concatenated to string 1 at parent end and then read back at the child end.

Logic: 2 pipes are used. One for the parent to write and child to read while another for the parent to read and child to write. Child writes a message to the write end of pipe 1 which is read by the parent from the read end of pipe 1. The parent concatenates the string read to the string it had set up and sends it back to the child by writing it to the write end of pipe 2 which is read by the child from the read end of pipe 2. The unused ends of both the pipes in both the processes are closed.

Code:

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

#define R_END 0
#define W_END 1

int main(int argc, char const *argv[])
{
    int fd1[2], fd2[2];
    int returnstatus1, returnstatus2;
    int pid;
    char pipe1msg[10] = "Mihir";
    char pipe2msg[10] = "Shri";
    char readmsg[10];

    returnstatus1 = pipe(fd1);

    if (returnstatus1 == -1)
    {
        printf("Unable to create Pipe 1\n");
        return 1;
    }

    returnstatus2 = pipe(fd2);

    if (returnstatus2 == -1)
    {
        printf("Unable to create Pipe 2\n");
```

```

        return 1;
    }

    pid = fork();

    if (pid > 0)
    {
        close(fd1[W_END]);
        close(fd2[R_END]);

        printf("Parent string is - %s\n", pipe1msg);

        read(fd1[R_END], readmsg, sizeof(readmsg));

        for (int i = 0; i < 4; i++)
            pipe1msg[i+5] = readmsg[i];

        write(fd2[W_END], pipe1msg, sizeof(pipe1msg)+1);
    }

    else
    {
        close(fd1[R_END]);
        close(fd2[W_END]);

        printf("Child writing to pipe 1. Message sent to parent is - %s\n", pipe2msg);

        write(fd1[W_END], pipe2msg, sizeof(pipe2msg)+1);

        read(fd2[R_END], readmsg, sizeof(readmsg));

        printf("Child read from pipe 2. Message received from parent is - %s\n",
readmsg);
    }

    return 0;
}

```


Output ss:

```
mihir@mihir-VirtualBox: ~/Desktop/OS/Lab/week5
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week5$ gcc string_concat.c
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week5$ ./a.out
Parent string is - Mihir
Child writing to pipe 1. Message sent to parent is - Shri
Child read from pipe 2. Message received from parent is - MihirShri
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week5$
```

(3) Substring generation at the child end of a string setup at the parent process end.

Logic: 2 pipes are used. One for the parent to write and child to read while another for the parent to read and child to write. Parent writes a message to the write end of pipe 1 which is read by the child from the read end of pipe 1. The child generates a substring and sends it to the parent by writing it to the write end of pipe 2 which is read by the parent from the read end of pipe 2.

The unused ends of both the pipes in both the processes are closed.

Code:

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

#define R_END 0
#define W_END 1

int main(int argc, char const *argv[])
{
    int fd1[2], fd2[2];
    int returnstatus1, returnstatus2;
    int pid;
    char pipe1msg[9] = "MihirShri";
    char pipe2msg[4];
    char readmsg[9];
    char readmsg2[4];
```

```

returnstatus1 = pipe(fd1);

if (returnstatus1 == -1)
{
    printf("Unable to create Pipe 1\n");
    return 1;
}

returnstatus2 = pipe(fd2);

if (returnstatus2 == -1)
{
    printf("Unable to create Pipe 2\n");
    return 1;
}

pid = fork();

if (pid > 0)
{
    close(fd1[R_END]);
    close(fd2[W_END]);

    printf("Parent writing to pipe 1. Message sent to child is - %s\n\n", pipe1msg);

    write(fd1[W_END], pipe1msg, sizeof(pipe1msg)+1);

    read(fd2[R_END], readmsg2, sizeof(readmsg2));

    printf("Parent read from pipe 2. Message read from child is - %s\n", readmsg2);
}

else
{
    close(fd1[W_END]);
    close(fd2[R_END]);

    read(fd1[R_END], readmsg, sizeof(readmsg));
}

```

```

    pipe2msg[0] = readmsg[4];
    pipe2msg[1] = readmsg[5];
    pipe2msg[2] = readmsg[6];

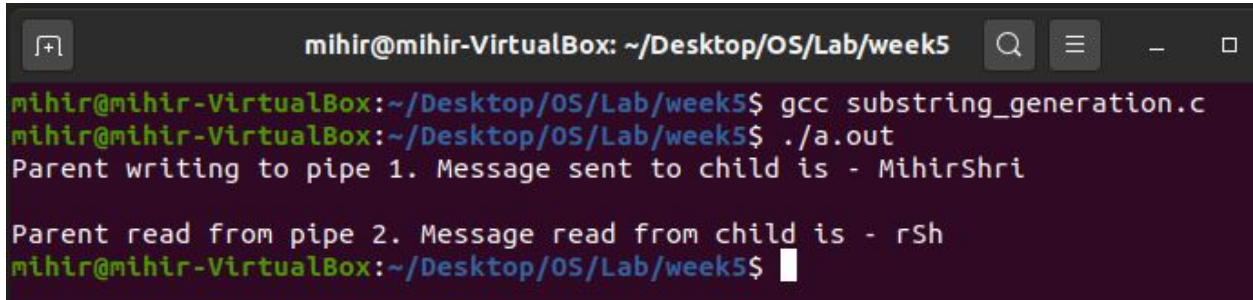
    write(fd2[W_END], pipe2msg, sizeof(pipe2msg)+1);

}

return 0;
}

```

Output ss:



```

mihir@mihir-VirtualBox: ~/Desktop/OS/Lab/week5
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week5$ gcc substring_generation.c
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week5$ ./a.out
Parent writing to pipe 1. Message sent to child is - MihirShri

Parent read from pipe 2. Message read from child is - rSh
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week5$

```

(4) String reversal and palindrome check using pipes / shared memory.

Logic: 2 pipes are used. One for the parent to write and child to read while another for the parent to read and child to write. Parent writes a message to the write end of pipe 1 which is read by the child from the read end of pipe 1. The child reverses the string and sends it back to the parent by writing it to the write end of pipe 2 which is read by the parent from the read end of pipe 2. The parent then performs a palindrome check on this string returned by the child.

The unused ends of both the pipes in both the processes are closed.

Code:

```

#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
#define RD_END 0
#define WR_END 1

```

```

int main(int argc, char const *argv[])
{
    int pipefd1[2],pipefd2[2];
    int status1, status2;
    char str1[9]="malayalaa"; //input string
    char str2[9];
    char str3[9];
    status1= pipe(pipefd1);
    if(status1== -1)
    {
        printf("Pipe Failed\n");
        return 0;
    }
    status2= pipe(pipefd2);
    if(status2== -1)
    {
        printf("Pipe Failed\n");
        return 0;
    }
    pid_t pid= fork();
    if(pid>0)
    {
        //parent
        close(pipefd1[RD_END]);
        close(pipefd2[WR_END]);
        printf("The original string by the parent is %s\n",str1);
        write(pipefd1[WR_END],str1,sizeof(str1)+1);
        read(pipefd2[RD_END],str2,sizeof(str2));
        printf("The reversed string by the child is %s\n",str2);
        int j=0,flag=1;
        for(int i=0;i<=8;i++)
        {
            if(str2[i]!=str1[i])
            {
                flag = 0;
                break;
            }
            j++;
        }
        if(flag==0)

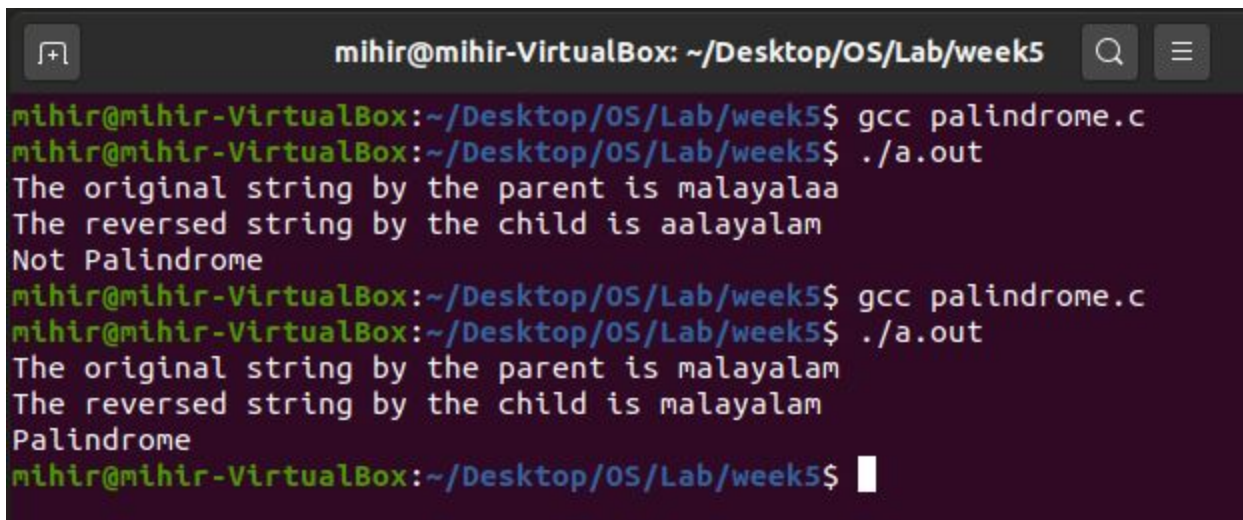
```

```

        {
            printf("Not Palindrome\n");
        }
        else
        {
            printf("Palindrome\n");
        }
    }
    else
    {
        //child
        close(pipefd1[WR_END]);
        close(pipefd2[RD_END]);
        read(pipefd1[RD_END],str2,sizeof(str2));
        int j=0;
        int i=sizeof(str2)/sizeof(str2[0]) -1;
        for(;i>=0;i--)
        {
            str3[j]=str2[i];
            j++;
        }
        write(pipefd2[WR_END],str3,sizeof(str3)+1);
    }
    return 0;
}

```

Output ss:



```

mihir@mihir-VirtualBox: ~/Desktop/OS/Lab/week5
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week5$ gcc palindrome.c
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week5$ ./a.out
The original string by the parent is malayalaa
The reversed string by the child is aalayalam
Not Palindrome
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week5$ gcc palindrome.c
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week5$ ./a.out
The original string by the parent is malayalam
The reversed string by the child is malayalam
Palindrome
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week5$

```

(5) Armstrong number generation within a range. The digit extraction, cubing can be the responsibility of the child while the checking for sum == no can happen in the child and the output list in the child.

Logic: As mentioned in the question, digit extraction and cubing is done by the parent (main) process without using any threads. An array of size 2 has been maintained whose first element is the original number while the second element is the sum of the cube of its digits. A runner function has been created and called using the `pthread_create()` function call. The array created above is passed as an argument to our thread function (runner function). The thread checks whether the first and second element of the array are equal or not (i.e. whether the number is equal to the sum of cubes of its digits or not) and prints the number if they are equal.

Code:

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

void *runner(void *param);

int main(int argc, char const *argv[])
{
    int n;
    pthread_t tid;
    pthread_attr_t attr;

    if (argc != 2)
    {
        fprintf(stderr, "Usage: ./a.out <integer value>\n");
        return -1;
    }

    n = atoi(argv[1]);

    if (n < 0)
    {
        fprintf(stderr, "%d must be >= 0\n", n);
        return -1;
    }
}
```

```

for (int i = 1; i <= n; i++)
{
    int *result = (int *)malloc(2*sizeof(int *));
    int num = i, d, cube = 0;

    while (num > 0)
    {
        d = num % 10;
        cube += d * d * d;
        num /= 10;
    }

    result[0] = i;
    result[1] = cube;

    pthread_attr_init(&attr);
    pthread_create(&tid, NULL, runner, (void *)result);
    pthread_join(tid, NULL);
    free(result);
}

return 0;
}

void *runner(void *param)
{
    int *result = (int *)param;

    if (result[0] == result[1])
        printf("%d\n", result[0]);

    pthread_exit(0);
}

```

Output ss:

```
mihir@mihir-VirtualBox: ~/Desktop/OS/Lab/week5
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week5$ gcc armstrong.c -lpthread
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week5$ ./a.out
Usage: ./a.out <integer value>
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week5$ ./a.out -2
-2 must be >= 0
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week5$ ./a.out 500
1
153
370
371
407
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week5$
```

(6) Ascending Order sort within Parent and Descending order sort (or vice versa) within the child process of an input array. (u can view as two different outputs –first entire array is ascending order sorted in op and then the second part descending order output).

Logic: The thread (runner function) is responsible for arranging the array elements in descending order while the parent process (main) arranges the elements in descending order.

Code:

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

```
void *runner(void *param);
```

```
int main(int argc, char const *argv[])
```



```

{
    int n;
    pthread_t tid;
    pthread_attr_t attr;

    if (argc < 2)
    {
        fprintf(stderr, "Usage: ./a.out <element_1> <element_2> <element_3>...\n");
        return -1;
    }

    n = argc;
    int *result = (int *)malloc((n+1) * sizeof(int *));
    result[0] = n;

    for(int i = 1; i < argc; i++)
        result[i] = atoi(argv[i]);

    pthread_attr_init(&attr);
    pthread_create(&tid, &attr, runner, (void *)result);
    pthread_join(tid, NULL);

    printf("Ascending Order: \n");

    for (int i = 1 ; i < n; i++)
    {
        for (int j = i + 1; j < n; ++j)
        {
            if(result[j] < result[i])
            {
                int t = result[i];
                result[i] = result[j];
                result[j] = t;
            }
        }
    }

    for(int i=1;i<n;i++)
        printf("%d ", result[i]);

```

```

printf("\n");

free(result);

return 0;
}

void *runner(void *param)
{
    int *result = (int *)param;
    int n = result[0];

    printf("Descending Order: \n");

    for(int i = 1; i < n; i++)
    {
        for (int j = i + 1; j < n; ++j)
        {
            if(result[i] < result[j])
            {
                int t = result[i];
                result[i] = result[j];
                result[j] = t;
            }
        }
    }

    for(int i=1;i<n;i++)
        printf("%d ",result[i]);

    printf("\n");

    pthread_exit(0);
}

```

Output ss:

```
mihir@mihir-VirtualBox: ~/Desktop/OS/Lab/week5
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week5$ gcc sort.c -lpthread
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week5$ ./a.out
Usage: ./a.out <element_1> <element_2> <element_3>...
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week5$ ./a.out 8 3 9 5 1
Descending Order:
9 8 5 3 1
Ascending Order:
1 3 5 8 9
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week5$
```

(7) Implement a multiprocessing version of binary search where the parent searches for the key in the first half and subsequent splits while the child searches in the other half of the array. By default you can implement a search for the first occurrence and later extend it to support multiple occurrences (duplicated elements search as well).

Logic: The thread (runner function0 is responsible for sorting and searching for the element in the second part of the array. While implementing binary search in the second part, $beg = n/2$, $end = n$. To tackle multiple occurrences of an element, the logic is, that since the array is sorted, all the duplicates of an element will be either towards the left of the element or towards the right of the element. The parent (main) function is responsible for doing the same (sorting and searching) in the second half of the array.

Code:

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

```
using namespace std;
```

```

void *runner(void *param);

int main(int argc, char const *argv[])
{
    pthread_t tid;
    pthread_attr_t attr;

    if (argc < 3)
    {
        fprintf(stderr, "Usage: ./a.out <key> <element_1> <element_2> ...\n");
        return -1;
    }

    int key = atoi(argv[1]);
    int n = argc - 2;
    int *result = (int *)malloc(n * sizeof(int *));

    result[0] = n;
    result[1] = key;

    for (int i = 2; i < argc; i++)
    {
        result[i] = atoi(argv[i]);
    }

    pthread_attr_init(&attr);
    pthread_create(&tid, &attr, runner, result);
    pthread_join(tid, NULL);

    for (int i = 0; i < n - 2; i++)
        result[i] = result[i+2];

    sort(result, result+n/2);

    int beg = 0;
    int end = n/2-1;
    int flag = 0;
    int c = 0;

    while (beg <= end)

```

```

{
    int mid = (beg+end) / 2;

    if (key == result[mid])
    {
        c++;

        for(int i = mid+1; i<= end; i++)
        {
            if(result[i] == key)
                c++;

            else
                break;
        }

        for(int i = mid-1; i >= beg; i--)
        {
            if(result[i] == key)
                c++;

            else
                break;
        }

        break;
    }

    else if (key < result[mid])
        end = mid - 1;

    else
        beg = mid + 1;
}

printf("Element %d found %d times in the first half.\n", key, c);

return 0;
}

```

```

void *runner(void *param)
{
    int *result = (int *)param;
    int n = result[0], key = result[1];
    int c = 0;

    for (int i = 0; i < n - 2; i++)
        result[i] = result[i+2];

    sort(result+n/2, result+n);

    int beg = n/2;
    int end = n-1;
    int flag = 0;

    while (beg <= end)
    {
        int mid = (beg+end) / 2;

        if (key == result[mid])
        {
            c++;

            for(int i = mid+1; i<= end; i++)
            {
                if(result[i] == key)
                    c++;

                else
                    break;
            }

            for(int i = mid-1; i >= beg; i--)
            {
                if(result[i] == key)
                    c++;

                else
                    break;
            }
        }
    }
}

```

```

        break;
    }

    else if (key < result[mid])
        end = mid - 1;

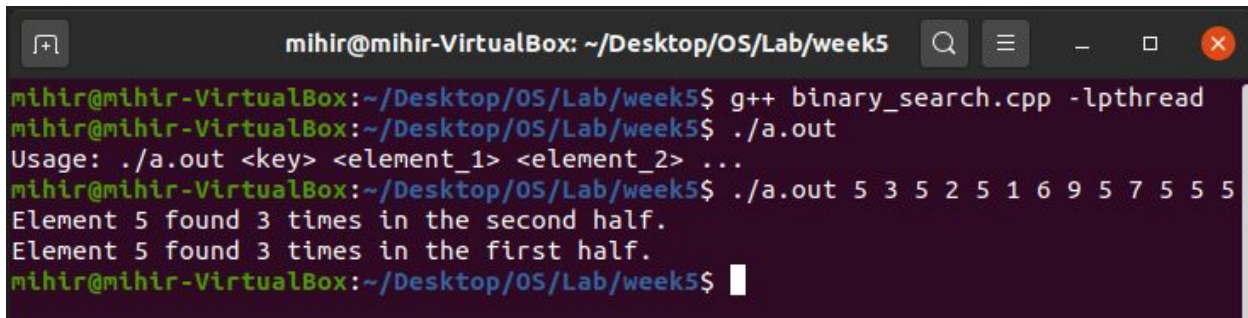
    else
        beg = mid + 1;
}

printf("Element %d found %d times in the second half.\n", key, c);

pthread_exit(0);
}

```

Output ss:



```

mihir@mihir-VirtualBox: ~/Desktop/OS/Lab/week5
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week5$ g++ binary_search.cpp -lpthread
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week5$ ./a.out
Usage: ./a.out <key> <element_1> <element_2> ...
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week5$ ./a.out 5 3 5 2 5 1 6 9 5 7 5 5
Element 5 found 3 times in the second half.
Element 5 found 3 times in the first half.
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week5$

```

(8) Read upon efficient ways of parallelizing the generation of Fibonacci series and apply the logic in a parent child relationship to contribute a faster version of fib series generation.

Logic: The Fibonacci numbers are the numbers in the following integer sequence. 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, In mathematical terms, the sequence F_n of Fibonacci numbers is defined by the recurrence relation $F_n = F_{n-1} + F_{n-2}$ with seed values $F_0 = 0$ and $F_1 = 1$. Now for calculating the $f(n-1)$ and $f(n-2)$, we can use 2 threads separately and add the returned values to calculate the $\text{fib}(n)$. The most

essential aspect is to observe that parallelization can help in reducing the time taken to generate the series. This is an effective setup in the logical wireframe.

Code:

```
#include<pthread.h>
#include<stdio.h>
#include<stdlib.h>
int fib=1,fib1=1,fib2=1;
void *gen1(void *param);
void *gen2(void *param);
int main(int argc, char const *argv[])
{
    pthread_t tid1,tid2;
    pthread_attr_t attr1,attr2;

    if (argc < 2)
    {
        fprintf(stderr, "Usage: ./a.out <n>\n");
        return -1;
    }

    printf("0\n");
    printf("1\n");
    int n=atoi(argv[1]);
    for(int i=2;i<=n;i++)
    {
        int *nums1 = (int*)malloc(sizeof(int));
        int *nums2 = (int*)malloc(sizeof(int));
        nums1[0]=i-1;nums2[0]=i-2;
        pthread_attr_init(&attr1);
        pthread_create(&tid1,NULL,gen1,(void *)nums1);
        pthread_attr_init(&attr2);
        pthread_create(&tid2,NULL,gen2,(void *)nums2);
        pthread_join(tid1,NULL);
        pthread_join(tid2,NULL);
        fib=fib1+fib2;
        printf("%d\n",fib);
    }
    return 0;
```



```

}
void *gen1(void *param)
{
    int * ar=(int *)param; int n = ar[0];
    int a = 0, b = 1, c, i;
        if( n == 0)
            fib1=a;
        else{
            for(i = 2; i <= n; i++)
            {
                c = a + b;
                a = b;
                b = c;
            }
            fib1=b;
        }
    pthread_exit(0);
}
void *gen2(void *param)
{
    int * ar=(int *)param; int n = ar[0];
    int a = 0, b = 1, c, i;
        if( n == 0)
            fib2=a;
        else{
            for(i = 2; i <= n; i++)
            {
                c = a + b;
                a = b;
                b = c;
            }
            fib2=b;
        }
    pthread_exit(0);
}

```

Output ss:

```
mihir@mihir-VirtualBox: ~/Desktop/OS/Lab/week5
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week5$ g++ fib.c -lpthread
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week5$ ./a.out
Usage: ./a.out <n>
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week5$ ./a.out 20
0
1
1
2
3
5
8
13
21
34
55
89
144
233
377
610
987
1597
2584
4181
6765
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week5$
```

(9) Longest Common Subsequence (Extra-Credits)

Logic: The code requires 2 aspects for generation of the longest common subsequence. The first aspect being initialization. I have used a runner function `gen1` for that purpose. It initializes the matrix for the base values of the substructure table. The next aspect is implementing the approach. The `gen` runner function is used to implement the approach as explained in the explanation above. The final string is stored in the global character array named `lcs`.

Code:

```
#include<pthread.h>
#include<stdio.h>
#include<stdlib.h>
#include<bits/stdc++.h>
using namespace std;
int L[100][100]; char lcs[100];
void *gen(void *param)
{
    char **argv= (char **)param;
    char *X = argv[1];
    char *Y = argv[2];
    int m=atoi(argv[3]); int n=atoi(argv[4]);
    // Following code is used to print LCS
    int index = L[m][n];
    //char lcs[index+1];
    lcs[index] = '\0';
    int i = m, j = n;
    while (i > 0 && j > 0)
    {
        if (X[i-1] == Y[j-1])
        {
            lcs[index-1] = X[i-1]; // Put current character in result
            i--; j--; index--;      // reduce values of i, j and index
        }
        else if (L[i-1][j] > L[i][j-1])
            i--;
        else
            j--;
    }

    pthread_exit(0);
}

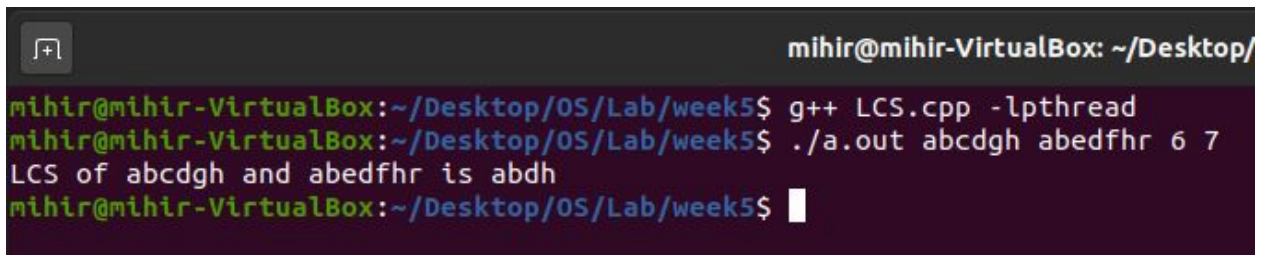
int main(int argc, char *argv[])
{
    memset(L,-1,sizeof(L));
    pthread_t tid;
    pthread_attr_t attr;
    int m= atoi(argv[argc-2]);
```

```

int n= atoi(argv[argc-1]);
char *X = argv[1];
char *Y = argv[2];
for (int i=0; i<=m; i++)
{
    for (int j=0; j<=n; j++)
    {
        if (i == 0 || j == 0)
            L[i][j] = 0;
        else if (X[i-1] == Y[j-1])
            L[i][j] = L[i-1][j-1] + 1;
        else
            L[i][j] = max(L[i-1][j], L[i][j-1]);
    }
}
pthread_attr_init(&attr);
pthread_create(&tid,NULL,gen,(void *)argv);
pthread_join(tid,NULL);
cout << "LCS of " << X << " and " << Y << " is " << lcs<<endl;
return 0;
}

```

Output ss:



```

mihir@mihir-VirtualBox: ~/Desktop/
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week5$ g++ LCS.cpp -lpthread
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week5$ ./a.out abcdgh abedfhr 6 7
LCS of abcdgh and abedfhr is abdh
mihir@mihir-VirtualBox:~/Desktop/OS/Lab/week5$

```