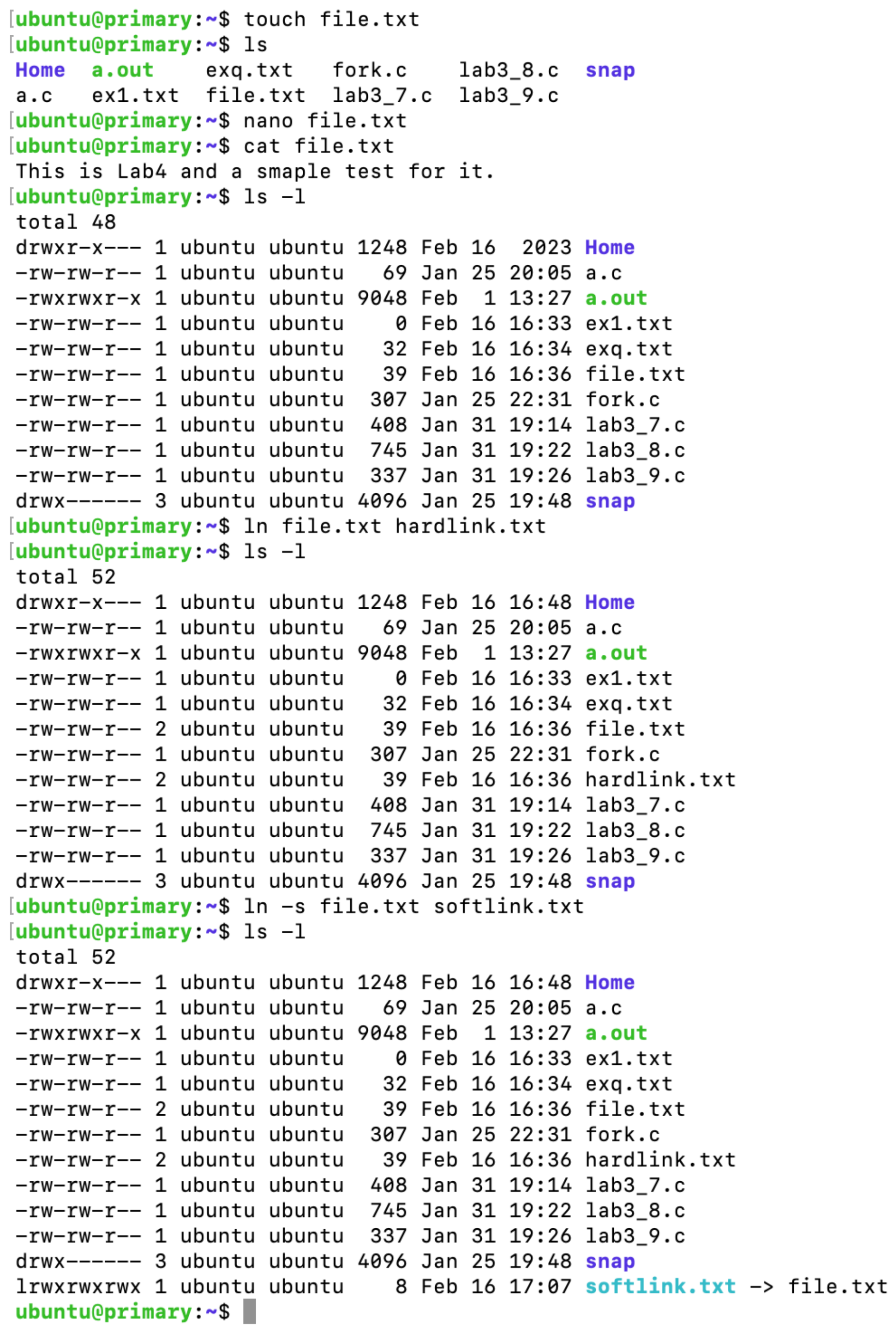
1. **Using the ln and ln -s commands, create hard and soft links.**

   Ans: below is the screen shot for all the required steps.



```
[ubuntu@primary:~$ touch file.txt
[ubuntu@primary:~$ ls
 Home   a.out     exq.txt   fork.c     lab3_8.c   snap
 a.c    ex1.txt   file.txt  lab3_7.c   lab3_9.c
[ubuntu@primary:~$ nano file.txt
[ubuntu@primary:~$ cat file.txt
 This is Lab4 and a smaple test for it.
[ubuntu@primary:~$ ls -l
 total 48
 drwxr-x--- 1 ubuntu ubuntu 1248 Feb 16  2023 Home
 -rw-rw-r-- 1 ubuntu ubuntu   69 Jan 25 20:05 a.c
 -rwxrwxr-x 1 ubuntu ubuntu 9048 Feb  1 13:27 a.out
 -rw-rw-r-- 1 ubuntu ubuntu    0 Feb 16 16:33 ex1.txt
 -rw-rw-r-- 1 ubuntu ubuntu   32 Feb 16 16:34 exq.txt
 -rw-rw-r-- 1 ubuntu ubuntu   39 Feb 16 16:36 file.txt
 -rw-rw-r-- 1 ubuntu ubuntu  307 Jan 25 22:31 fork.c
 -rw-rw-r-- 1 ubuntu ubuntu  408 Jan 31 19:14 lab3_7.c
 -rw-rw-r-- 1 ubuntu ubuntu  745 Jan 31 19:22 lab3_8.c
 -rw-rw-r-- 1 ubuntu ubuntu  337 Jan 31 19:26 lab3_9.c
 drwx------ 3 ubuntu ubuntu 4096 Jan 25 19:48 snap
[ubuntu@primary:~$ ln file.txt hardlink.txt
[ubuntu@primary:~$ ls -l
 total 52
 drwxr-x--- 1 ubuntu ubuntu 1248 Feb 16 16:48 Home
 -rw-rw-r-- 1 ubuntu ubuntu   69 Jan 25 20:05 a.c
 -rwxrwxr-x 1 ubuntu ubuntu 9048 Feb  1 13:27 a.out
 -rw-rw-r-- 1 ubuntu ubuntu    0 Feb 16 16:33 ex1.txt
 -rw-rw-r-- 1 ubuntu ubuntu   32 Feb 16 16:34 exq.txt
 -rw-rw-r-- 2 ubuntu ubuntu   39 Feb 16 16:36 file.txt
 -rw-rw-r-- 1 ubuntu ubuntu  307 Jan 25 22:31 fork.c
 -rw-rw-r-- 2 ubuntu ubuntu   39 Feb 16 16:36 hardlink.txt
 -rw-rw-r-- 1 ubuntu ubuntu  408 Jan 31 19:14 lab3_7.c
 -rw-rw-r-- 1 ubuntu ubuntu  745 Jan 31 19:22 lab3_8.c
 -rw-rw-r-- 1 ubuntu ubuntu  337 Jan 31 19:26 lab3_9.c
 drwx------ 3 ubuntu ubuntu 4096 Jan 25 19:48 snap
[ubuntu@primary:~$ ln -s file.txt softlink.txt
[ubuntu@primary:~$ ls -l
 total 52
 drwxr-x--- 1 ubuntu ubuntu 1248 Feb 16 16:48 Home
 -rw-rw-r-- 1 ubuntu ubuntu   69 Jan 25 20:05 a.c
 -rwxrwxr-x 1 ubuntu ubuntu 9048 Feb  1 13:27 a.out
 -rw-rw-r-- 1 ubuntu ubuntu    0 Feb 16 16:33 ex1.txt
 -rw-rw-r-- 1 ubuntu ubuntu   32 Feb 16 16:34 exq.txt
 -rw-rw-r-- 2 ubuntu ubuntu   39 Feb 16 16:36 file.txt
 -rw-rw-r-- 1 ubuntu ubuntu  307 Jan 25 22:31 fork.c
 -rw-rw-r-- 2 ubuntu ubuntu   39 Feb 16 16:36 hardlink.txt
 -rw-rw-r-- 1 ubuntu ubuntu  408 Jan 31 19:14 lab3_7.c
 -rw-rw-r-- 1 ubuntu ubuntu  745 Jan 31 19:22 lab3_8.c
 -rw-rw-r-- 1 ubuntu ubuntu  337 Jan 31 19:26 lab3_9.c
 drwx------ 3 ubuntu ubuntu 4096 Jan 25 19:48 snap
 lrwxrwxrwx 1 ubuntu ubuntu    8 Feb 16 17:07 softlink.txt -> file.txt
 ubuntu@primary:~$
```

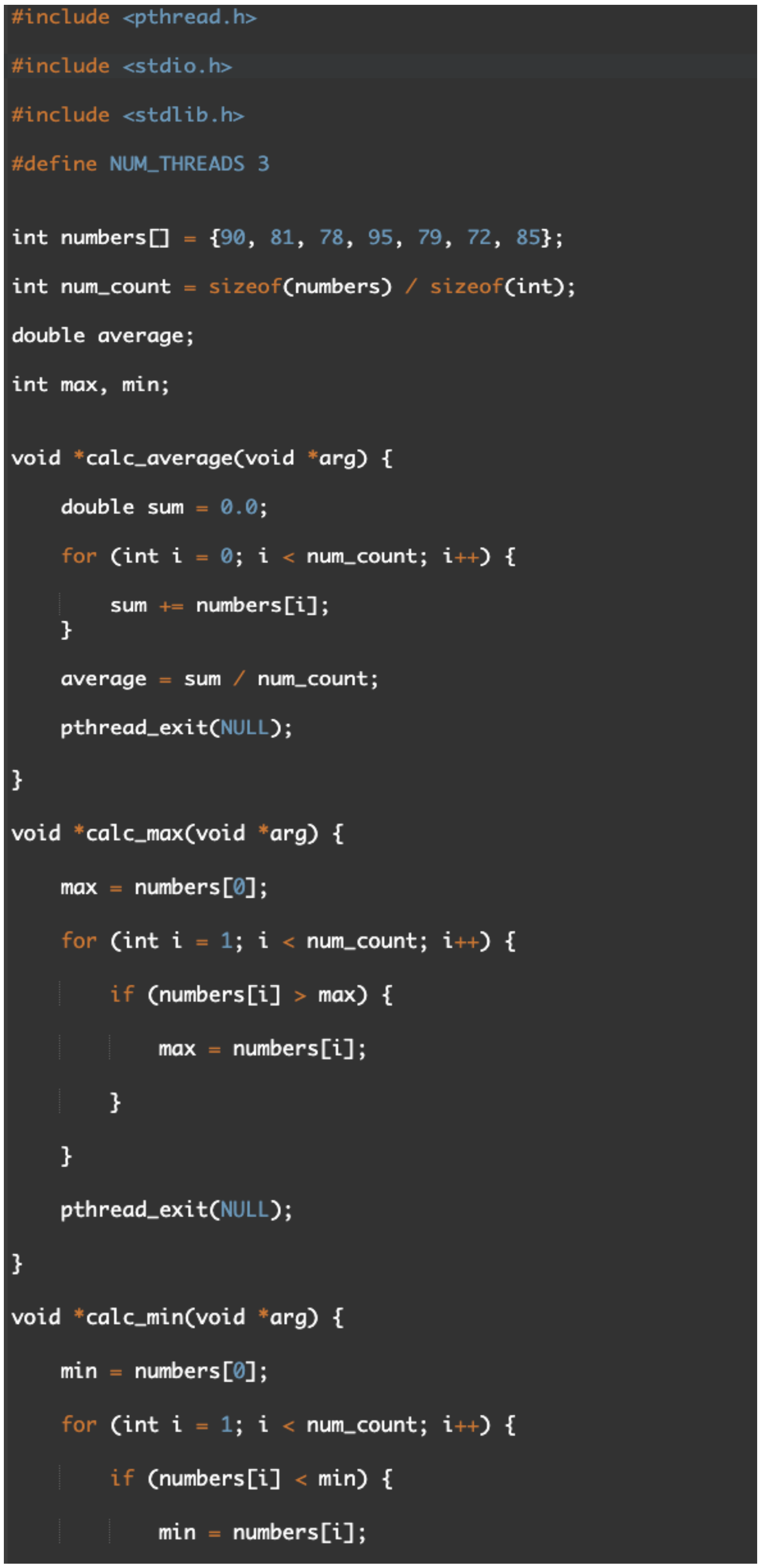
2. Create a multithreaded program that computes different statistical values for a set of numbers. When given a series of numbers on the command line, this application will start two independent worker threads. One thread will compute the greatest value, and the next will compute the minimum value. Assume your program is given a list of integers. (The array of numbers must be provided as a parameter to the threads, and the thread must return the calculated value to the main thread.)

2	**20**	**25**	**5**	**70**	**90**	**98**

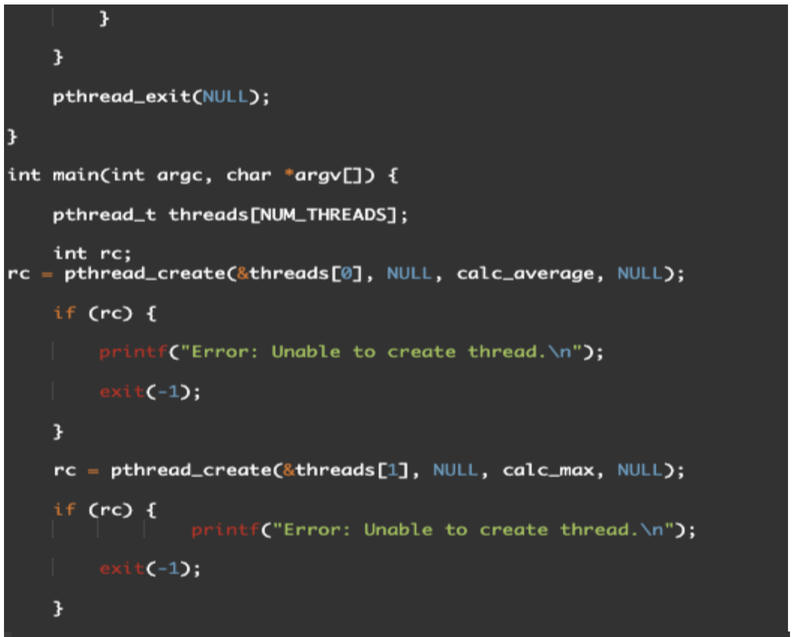
Ans:

The given given in the screenshots below demonstrates how to use pthreads (POSIX threads) to perform multi-threaded programming. The program creates three threads, each of which performs a specific task: calculating the average, maximum, and minimum values of a set of integers stored in an array. The threads execute their respective functions concurrently, and the main thread waits for all of them to complete using the pthread_join() function. Once all threads have finished their tasks, the main thread prints out the calculated values of the average, maximum, and minimum values. The program demonstrates how to use pthreads to perform parallel processing and improve program performance by distributing tasks across multiple threads.

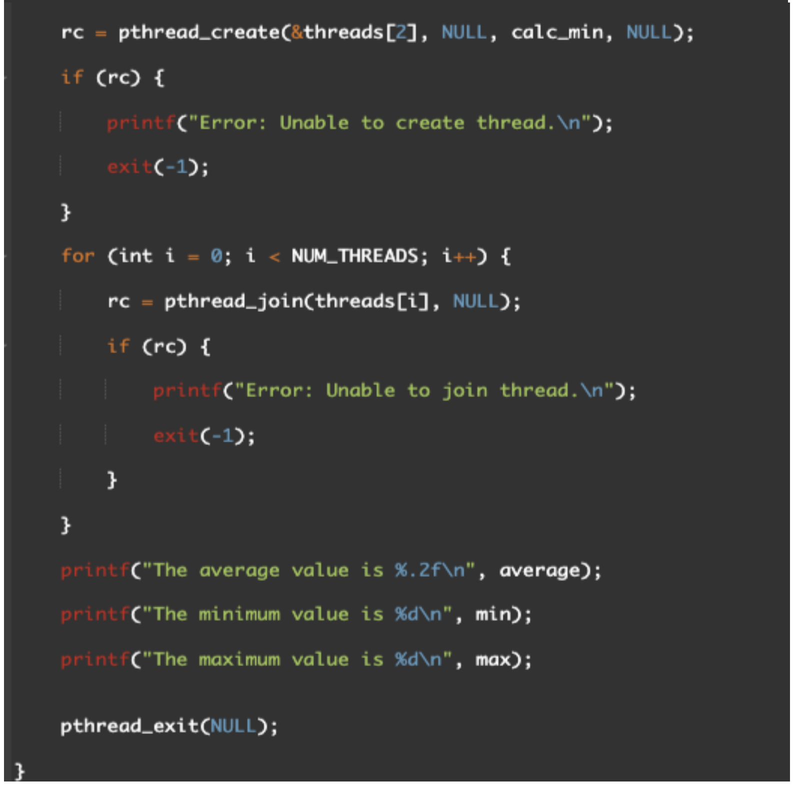
Code:



```c
#include <pthread.h>

#include <stdio.h>

#include <stdlib.h>

#define NUM_THREADS 3

int numbers[] = {90, 81, 78, 95, 79, 72, 85};

int num_count = sizeof(numbers) / sizeof(int);

double average;

int max, min;

void *calc_average(void *arg) {

    double sum = 0.0;

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

        sum += numbers[i];
    }

    average = sum / num_count;

    pthread_exit(NULL);

}

void *calc_max(void *arg) {

    max = numbers[0];

    for (int i = 1; i < num_count; i++) {

        if (numbers[i] > max) {

            max = numbers[i];

        }

    }

    pthread_exit(NULL);

}

void *calc_min(void *arg) {

    min = numbers[0];

    for (int i = 1; i < num_count; i++) {

        if (numbers[i] < min) {

            min = numbers[i];
```

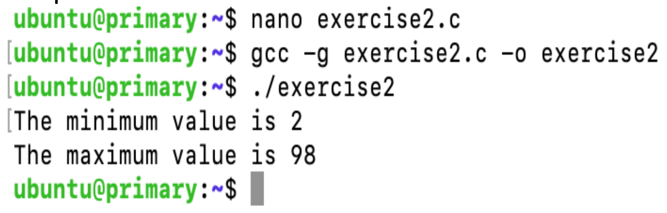


```
        }
    }
    pthread_exit(NULL);
}
int main(int argc, char *argv[]) {
    pthread_t threads[NUM_THREADS];
    int rc;
rc = pthread_create(&threads[0], NULL, calc_average, NULL);
    if (rc) {
        printf("Error: Unable to create thread.\n");
        exit(-1);
    }
    rc = pthread_create(&threads[1], NULL, calc_max, NULL);
    if (rc) {
                printf("Error: Unable to create thread.\n");
        exit(-1);
    }
```



```
    rc = pthread_create(&threads[2], NULL, calc_min, NULL);
    if (rc) {
        printf("Error: Unable to create thread.\n");
        exit(-1);
    }
    for (int i = 0; i < NUM_THREADS; i++) {
        rc = pthread_join(threads[i], NULL);
        if (rc) {
            printf("Error: Unable to join thread.\n");
            exit(-1);
        }
    }
    printf("The average value is %.2f\n", average);
    printf("The minimum value is %d\n", min);
    printf("The maximum value is %d\n", max);

    pthread_exit(NULL);
}
```

Output:

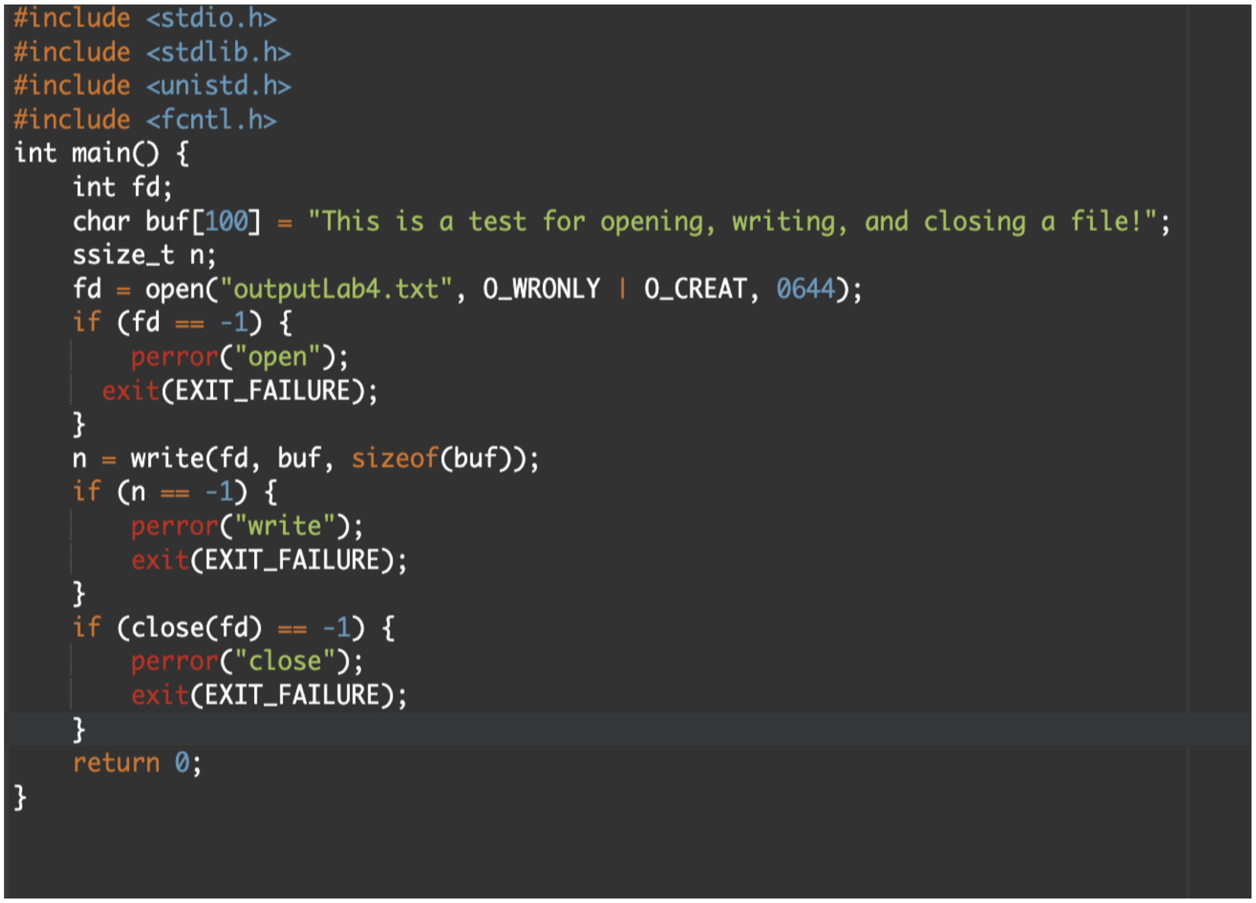


```
ubuntu@primary:~$ nano exercise2.c
ubuntu@primary:~$ gcc -g exercise2.c -o exercise2
ubuntu@primary:~$ ./exercise2
The minimum value is 2
The maximum value is 98
ubuntu@primary:~$
```

3. Write a C program that opens the file "outputLab4.txt" for writing and appends the phrase "This is a test for opening, writing, and closing a file!"

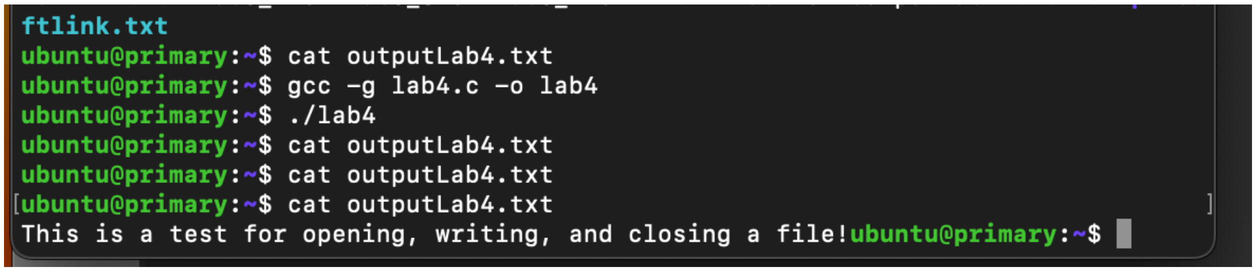
Ans:

The code in the below screenshot demonstrates how to create, write to, and close a file using system calls in C. The code first declares a file descriptor variable 'fd' to hold the file descriptor of the file being created. A buffer 'buf' is initialized with some text to be written to the file. The code then uses the 'open' function to create a file named "outputLab4.txt" with write-only access and set file permissions to 0644. If the 'open' call fails, the code prints an error message and exits the program. The 'write' function is then used to write the content of the buffer 'buf' to the file. If the 'write' call fails, the code prints an error message and exits the program. Finally, the file is closed using the 'close' function. If the 'close' call fails, the code prints an error message and exits the program.



```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
int main() {
    int fd;
    char buf[100] = "This is a test for opening, writing, and closing a file!";
    ssize_t n;
    fd = open("outputLab4.txt", O_WRONLY | O_CREAT, 0644);
    if (fd == -1) {
        perror("open");
      exit(EXIT_FAILURE);
    }
    n = write(fd, buf, sizeof(buf));
    if (n == -1) {
        perror("write");
        exit(EXIT_FAILURE);
    }
    if (close(fd) == -1) {
        perror("close");
        exit(EXIT_FAILURE);
    }
    return 0;
}
```

Output:



```
ftlink.txt
ubuntu@primary:~$ cat outputLab4.txt
ubuntu@primary:~$ gcc -g lab4.c -o lab4
ubuntu@primary:~$ ./lab4
ubuntu@primary:~$ cat outputLab4.txt
ubuntu@primary:~$ cat outputLab4.txt
ubuntu@primary:~$ cat outputLab4.txt
This is a test for opening, writing, and closing a file!ubuntu@primary:~$
```

4. Write a program for matrix addition, subtraction and multiplication using multithreading.

Ans:

The code given screenshot given below defines three functions for performing matrix operations of addition, subtraction, and multiplication using threads in parallel. The input matrix A and B are taken from the user, and the resultant matrix is stored in resultMatrix. The main function initializes two threads to perform subtraction and multiplication operations using the functions defined above. The threads are created using pthread_t, and the arguments are passed using void pointers. The number of rows in matrix A is divided into two parts for the threads to work on. The threads execute their respective functions in parallel and update the resultant matrix. Finally, the threads are joined using pthread_join and the resultant matrix is printed to the console.

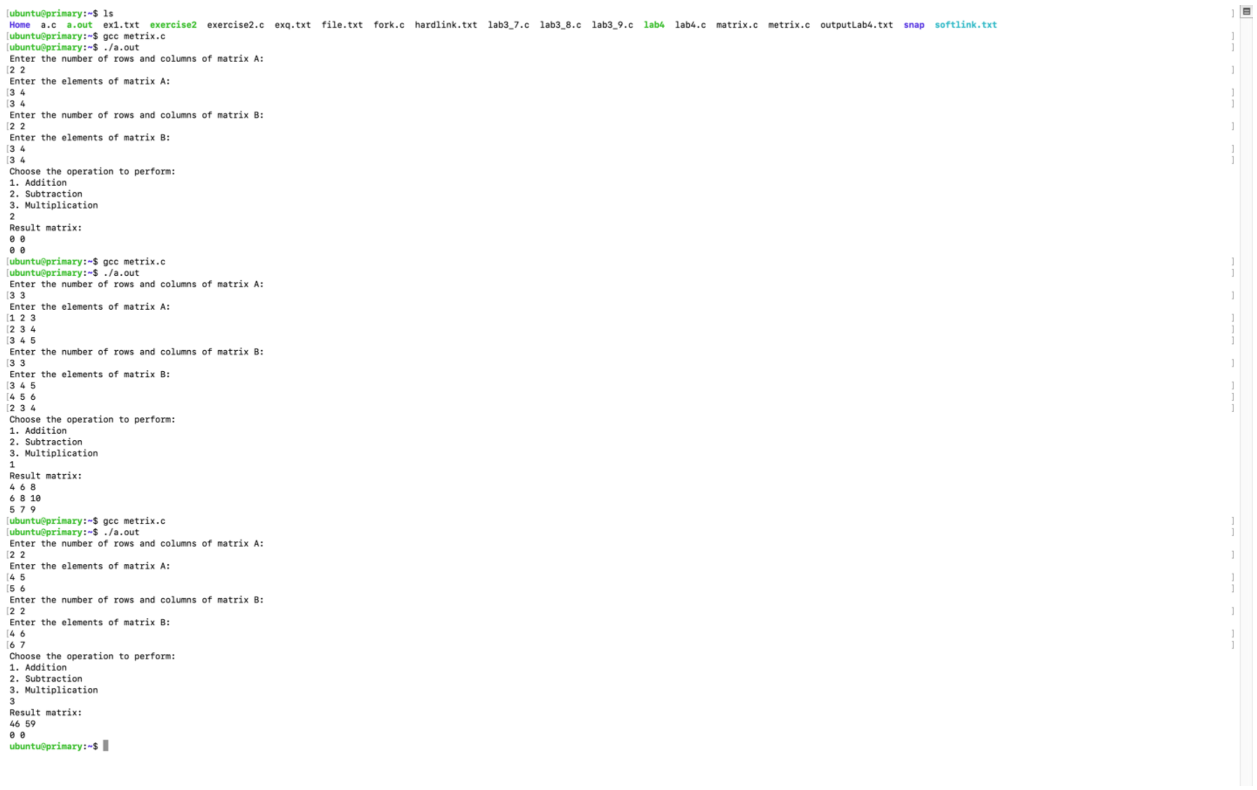
Code:

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

#define MAX_SIZE 10

int matrixA[MAX_SIZE][MAX_SIZE], matrixB[MAX_SIZE][MAX_SIZE], resultMatrix[MAX_SIZE][MAX_SIZE];
int rowsA, colsA, rowsB, colsB, operation;

void *addition(void *arg)
{
    int i, j;
    for (i = 0; i < rowsA; i++) {
        for (j = 0; j < colsA; j++) {
            resultMatrix[i][j] = matrixA[i][j] + matrixB[i][j];
        }
    }
    pthread_exit(NULL);
}

void *subtraction(void *arg)
{
    int i, j;
    int start_row = *((int*)arg);
    int end_row = start_row + (rowsA / 2);
    for (i = start_row; i < end_row; i++) {
        for (j = 0; j < colsA; j++) {
            resultMatrix[i][j] = matrixA[i][j] - matrixB[i][j];
        }
    }
    pthread_exit(NULL);
}

void *multiplication(void *arg)
{
    int i, j, k;
    int start_row = *((int*)arg);
    int end_row = start_row + (rowsA / 2);
    for (i = start_row; i < end_row; i++) {
        for (j = 0; j < colsB; j++) {
            resultMatrix[i][j] = 0;
            for (k = 0; k < colsA; k++) {
                resultMatrix[i][j] += matrixA[i][k] * matrixB[k][j];
            }
        }
    }
    pthread_exit(NULL);
}
```

```
int main()
{
    int i, j;
    pthread_t tid1, tid2;
    int start_row1 = 0, start_row2 = rowsA / 2;

    // Take input from user
    printf("Enter the number of rows and columns of matrix A:\n");
    scanf("%d %d", &rowsA, &colsA);

    printf("Enter the elements of matrix A:\n");
    for (i = 0; i < rowsA; i++) {
        for (j = 0; j < colsA; j++) {
            scanf("%d", &matrixA[i][j]);
        }
    }

    printf("Enter the number of rows and columns of matrix B:\n");
    scanf("%d %d", &rowsB, &colsB);

    printf("Enter the elements of matrix B:\n");
    for (i = 0; i < rowsB; i++) {
        for (j = 0; j < colsB; j++) {
            scanf("%d", &matrixB[i][j]);
        }
    }

    // Choose operation
    printf("Choose the operation to perform:\n");
    printf("1. Addition\n");
    printf("2. Subtraction\n");
    printf("3. Multiplication\n");
    scanf("%d", &operation);

    // Check if operation is valid


    switch (operation) {
        case 1:
        if (rowsA != rowsB || colsA != colsB) {
            printf("Invalid operation: matrix dimensions are not compatible.\n");
            return 1;
        }
        pthread_create(&tid1, NULL, addition, NULL);
        break;
        case 2:
        if (rowsA != rowsB || colsA != colsB) {
            printf("Invalid operation: matrix dimensions are not compatible.\n");
            return 1;
        }
        pthread_create(&tid1, NULL, subtraction, &start_row1);
        pthread_create(&tid2, NULL, subtraction, &start_row2);
```

```c
        break;
        case 3:
        if (colsA != rowsB) {
            printf("Invalid operation: matrix dimensions are not compatible.\n");
            return 1;
        }
        pthread_create(&tid1, NULL, multiplication, &start_row1);
        pthread_create(&tid2, NULL, multiplication, &start_row2);
        break;
        default:
        printf("Invalid operation.\n");
        return 1;
    }
    // Wait for threads to finish
    if (operation == 1) {
        pthread_join(tid1, NULL);
    } else {
        pthread_join(tid1, NULL);
        pthread_join(tid2, NULL);
    }

// Output result matrix
    printf("Result matrix:\n");
    for (i = 0; i < rowsA; i++) {
        for (j = 0; j < colsB; j++) {
            printf("%d ", resultMatrix[i][j]);
        }
     printf("\n");
    }

    return 0;
}
```

Screenshots:



```
ubuntu@primary:~$ ls
Home  a.c  a.out  ex1.txt  exercise2  exercise2.c  exq.txt  file.txt  fork.c  hardlink.txt  lab3_7.c  lab3_8.c  lab3_9.c  lab4  lab4.c  matrix.c  metrix.c  outputLab4.txt  snap  softlink.txt
ubuntu@primary:~$ gcc metrix.c
ubuntu@primary:~$ ./a.out
Enter the number of rows and columns of matrix A:
2 2
Enter the elements of matrix A:
3 4
3 4
Enter the number of rows and columns of matrix B:
2 2
Enter the elements of matrix B:
3 4
3 4
Choose the operation to perform:
1. Addition
2. Subtraction
3. Multiplication
2
Result matrix:
0 0
0 0
ubuntu@primary:~$ gcc metrix.c
ubuntu@primary:~$ ./a.out
Enter the number of rows and columns of matrix A:
3 3
Enter the elements of matrix A:
1 2 3
2 3 4
3 4 5
Enter the number of rows and columns of matrix B:
3 3
Enter the elements of matrix B:
3 4 5
4 5 6
2 3 4
Choose the operation to perform:
1. Addition
2. Subtraction
3. Multiplication
1
Result matrix:
4 6 8
6 8 10
5 7 9
ubuntu@primary:~$ gcc metrix.c
ubuntu@primary:~$ ./a.out
Enter the number of rows and columns of matrix A:
2 2
Enter the elements of matrix A:
4 5
5 6
Enter the number of rows and columns of matrix B:
2 2
Enter the elements of matrix B:
4 6
6 7
Choose the operation to perform:
1. Addition
2. Subtraction
3. Multiplication
3
Result matrix:
46 59
0 0
ubuntu@primary:~$
```