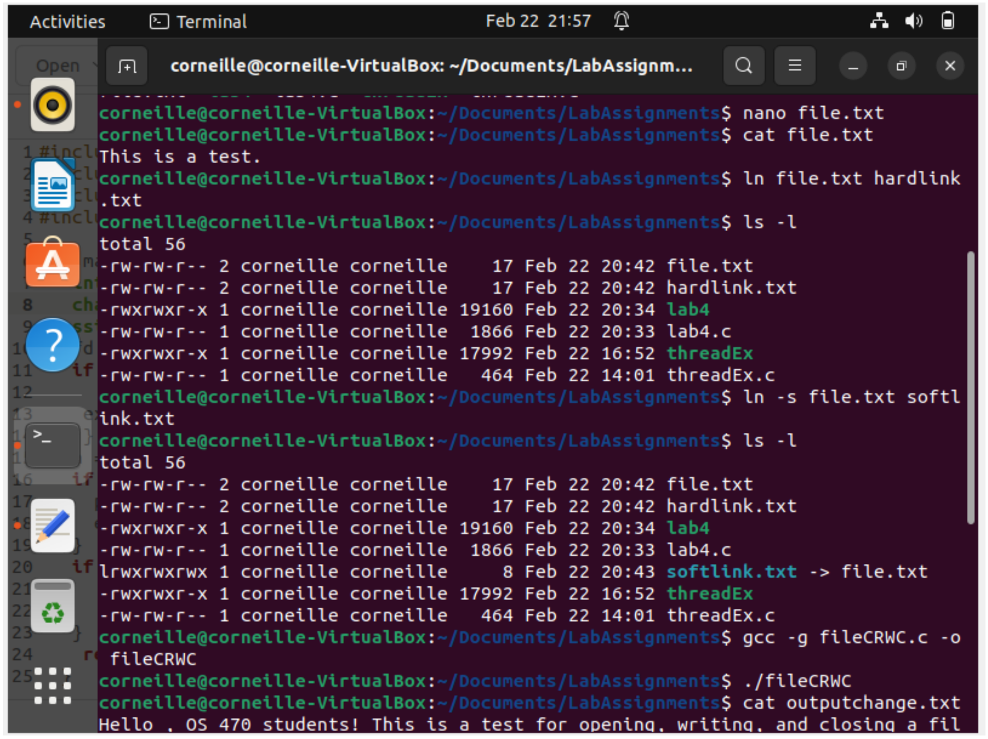
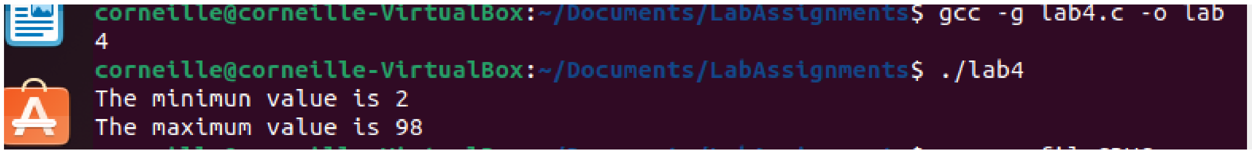
Lab4: Creating Threads in C
Linux System Calls for File and Directory Management

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



```
corneille@corneille-VirtualBox:~/Documents/LabAssignments$ nano file.txt
corneille@corneille-VirtualBox:~/Documents/LabAssignments$ cat file.txt
This is a test.
corneille@corneille-VirtualBox:~/Documents/LabAssignments$ ln file.txt hardlink
.txt
corneille@corneille-VirtualBox:~/Documents/LabAssignments$ ls -l
total 56
-rw-rw-r-- 2 corneille corneille    17 Feb 22 20:42 file.txt
-rw-rw-r-- 2 corneille corneille    17 Feb 22 20:42 hardlink.txt
-rwxrwxr-x 1 corneille corneille 19160 Feb 22 20:34 lab4
-rw-rw-r-- 1 corneille corneille  1866 Feb 22 20:33 lab4.c
-rwxrwxr-x 1 corneille corneille 17992 Feb 22 16:52 threadEx
-rw-rw-r-- 1 corneille corneille   464 Feb 22 14:01 threadEx.c
corneille@corneille-VirtualBox:~/Documents/LabAssignments$ ln -s file.txt softl
ink.txt
corneille@corneille-VirtualBox:~/Documents/LabAssignments$ ls -l
total 56
-rw-rw-r-- 2 corneille corneille    17 Feb 22 20:42 file.txt
-rw-rw-r-- 2 corneille corneille    17 Feb 22 20:42 hardlink.txt
-rwxrwxr-x 1 corneille corneille 19160 Feb 22 20:34 lab4
-rw-rw-r-- 1 corneille corneille  1866 Feb 22 20:33 lab4.c
lrwxrwxrwx 1 corneille corneille     8 Feb 22 20:43 softlink.txt -> file.txt
-rwxrwxr-x 1 corneille corneille 17992 Feb 22 16:52 threadEx
-rw-rw-r-- 1 corneille corneille   464 Feb 22 14:01 threadEx.c
corneille@corneille-VirtualBox:~/Documents/LabAssignments$ gcc -g fileCRWC.c -o
 fileCRWC
corneille@corneille-VirtualBox:~/Documents/LabAssignments$ ./fileCRWC
corneille@corneille-VirtualBox:~/Documents/LabAssignments$ cat outputchange.txt
Hello , OS 470 students! This is a test for opening, writing, and closing a fil
```

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



```
corneille@corneille-VirtualBox:~/Documents/LabAssignments$ gcc -g lab4.c -o lab
4
corneille@corneille-VirtualBox:~/Documents/LabAssignments$ ./lab4
The minimun value is 2
The maximum value is 98
```

Here is the code:

```
#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 minimum value is %d\n", min);
    printf("The maximum value is %d\n", max);

    pthread_exit(NULL);
}
```
The program creates three threads that perform calculations on an array of numbers. Specifically, the threads calculate and print the maximum and minimum values of the numbers.

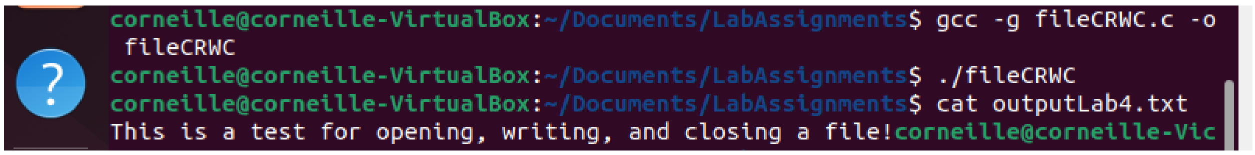
3. **Write a C program that opens the file "outputLab4.txt" for writing and appends the phrase "This is a test for opening, writing, and closing a file!"**

Here is the code:

```
#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;
}
```

The program creates a file named "outputLab4.txt" and writes the string " This is a test for opening, writing, and closing a file!" to it. The **open**, **write**, and **close** functions are used for these operations.

Here is the output:



```
corneille@corneille-VirtualBox:~/Documents/LabAssignments$ gcc -g fileCRWC.c -o
 fileCRWC
corneille@corneille-VirtualBox:~/Documents/LabAssignments$ ./fileCRWC
corneille@corneille-VirtualBox:~/Documents/LabAssignments$ cat outputLab4.txt
This is a test for opening, writing, and closing a file!corneille@corneille-Vic
```

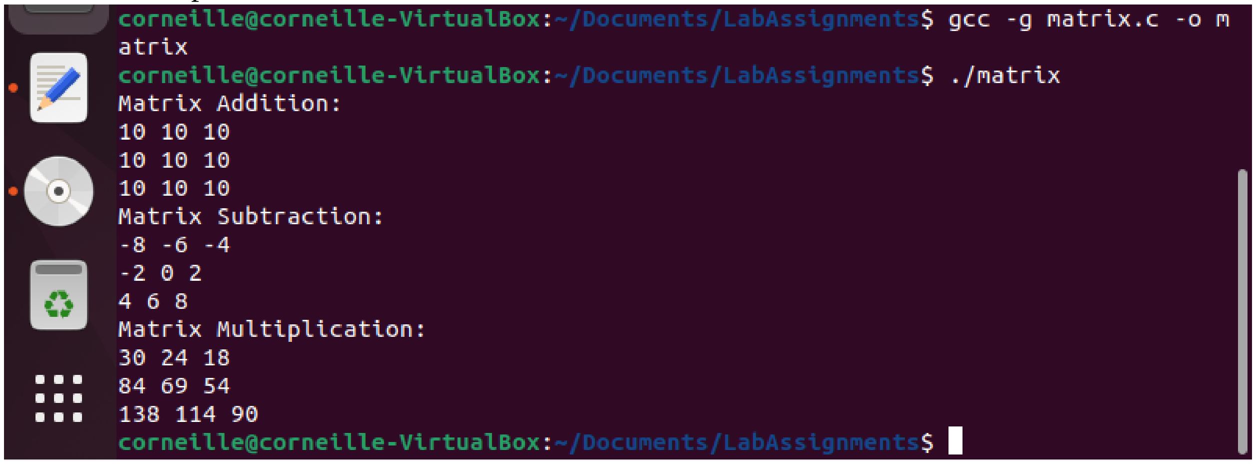
4. **Write a program for matrix addition, subtraction and multiplication using multithreading:**

Here is the code

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

#define ROWS 3
#define COLS 3

int matrix1[ROWS][COLS] = { {1, 2, 3}, {4, 5, 6}, {7, 8, 9} };
int matrix2[ROWS][COLS] = { {9, 8, 7}, {6, 5, 4}, {3, 2, 1} };
int result_addition[ROWS][COLS];
int result_subtraction[ROWS][COLS];
int result_multiplication[ROWS][COLS];

// Function to perform matrix addition
void *addition(void *arg) {
   int i, j;
   for(i=0; i<ROWS; i++) {
      for(j=0; j<COLS; j++) {
         result_addition[i][j] = matrix1[i][j] + matrix2[i][j];
      }
   }
   pthread_exit(NULL);
}

// Function to perform matrix subtraction
void *subtraction(void *arg) {
   int i, j;
   for(i=0; i<ROWS; i++) {
      for(j=0; j<COLS; j++) {
         result_subtraction[i][j] = matrix1[i][j] - matrix2[i][j];
      }
   }
   pthread_exit(NULL);
}

// Function to perform matrix multiplication
void *multiplication(void *arg) {
   int i, j, k;
   for(i=0; i<ROWS; i++) {
        for(j=0; j<COLS; j++) {
            result_multiplication[i][j] = 0;
            for(k=0; k<COLS; k++) {
                result_multiplication[i][j] += matrix1[i][k] * matrix2[k][j];
            }
        }
    }
    pthread_exit(NULL);
}

int main() {
    pthread_t tid_add, tid_sub, tid_mul;

    // Create threads to perform matrix addition, subtraction, and multiplication
    pthread_create(&tid_add, NULL, addition, NULL);
    pthread_create(&tid_sub, NULL, subtraction, NULL);
    pthread_create(&tid_mul, NULL, multiplication, NULL);

    // Wait for all threads to finish
    pthread_join(tid_add, NULL);
    pthread_join(tid_sub, NULL);
    pthread_join(tid_mul, NULL);

    // Print the results
    printf("Matrix Addition:\n");
    for(int i=0; i<ROWS; i++) {
        for(int j=0; j<COLS; j++) {
            printf("%d ", result_addition[i][j]);
        }
        printf("\n");
    }

    printf("Matrix Subtraction:\n");
    for(int i=0; i<ROWS; i++) {
        for(int j=0; j<COLS; j++) {
            printf("%d ", result_subtraction[i][j]);
        }
        printf("\n");
    }

    printf("Matrix Multiplication:\n");
    for(int i=0; i<ROWS; i++) {
        for(int j=0; j<COLS; j++) {
            printf("%d ", result_multiplication[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

The program performs matrix addition, subtraction, and multiplication using multithreading in C to run in Linux VirtualBox. It defines matrices and functions to perform the operations, creates threads to run each function, and uses **pthread_join()** to wait for each thread to finish before printing the results.

Here is the output



```
corneille@corneille-VirtualBox:~/Documents/LabAssignments$ gcc -g matrix.c -o m
atrix
corneille@corneille-VirtualBox:~/Documents/LabAssignments$ ./matrix
Matrix Addition:
10 10 10
10 10 10
10 10 10
Matrix Subtraction:
-8 -6 -4
-2 0 2
4 6 8
Matrix Multiplication:
30 24 18
84 69 54
138 114 90
corneille@corneille-VirtualBox:~/Documents/LabAssignments$
```