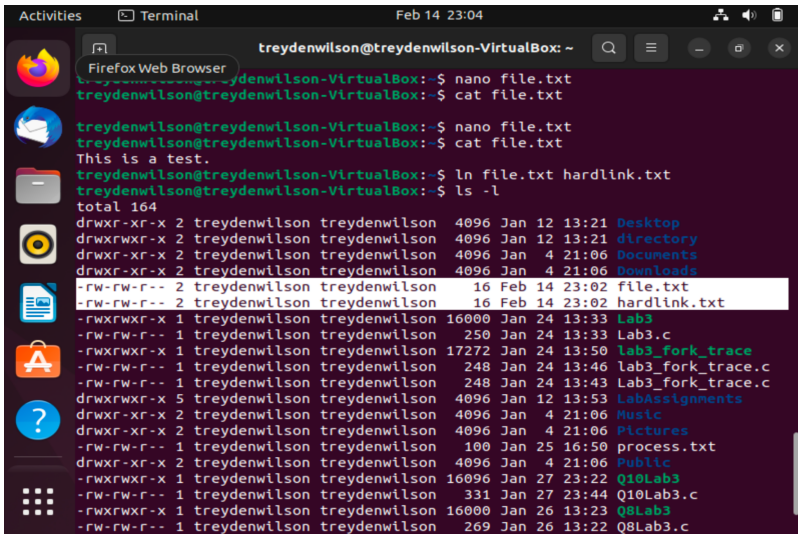
Treyden Wilson
Lab 4
CS 470 Operating Systems

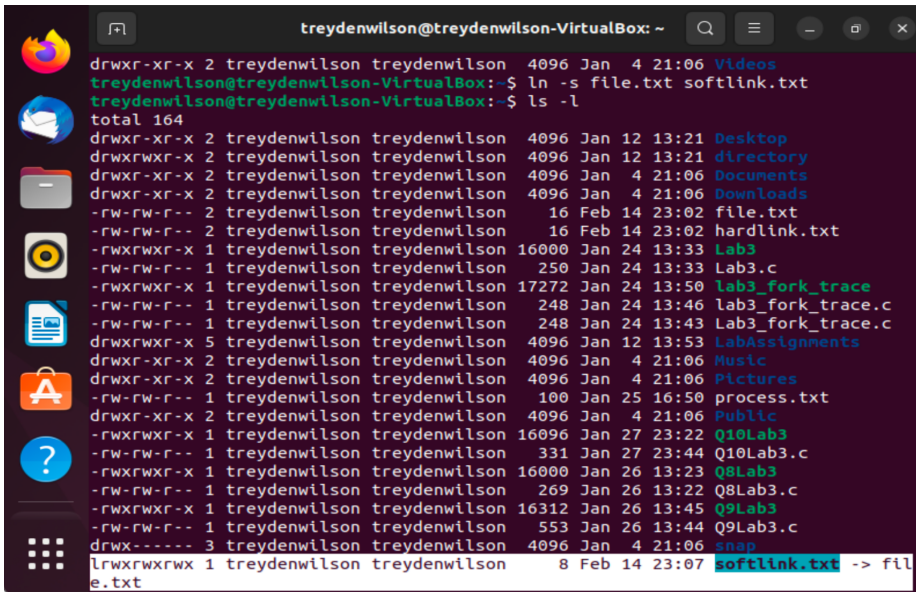
1. **Using the ln and ln-s commands, create hard and soft links.**

Hard Link:



The following code creates a hard link between file.txt and hardlink.txt. Furthermore, it shows that the original file and hard link have two hard links pointing to them. Finally, we verified that the hard link is indeed there, and the above screenshot proves that.

Soft Link:



The following code creates a soft link using the ln -s command with file.txt and softlink.txt. Then ls -l is used to view the soft link, which is shown at the bottom of the screenshot. You can see the soft link highlighted in white above.

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.)

```c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#define NUM_THREADS 3

int numbers[] = {2, 20, 25, 5, 70, 90, 98};

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];

}

}

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);

}
```

```
treydenwilson@treydenwilson-VirtualBox:~$ gedit
treydenwilson@treydenwilson-VirtualBox:~$ gcc Q2Lab4.c -o Q2Lab4
treydenwilson@treydenwilson-VirtualBox:~$ ./Q2Lab4
The average value is 44.29
The minimum value is 2
The maximum value is 98
treydenwilson@treydenwilson-VirtualBox:~$
```

The following code computes different statistical values of a set of numbers. One thread computes the greatest max value, another thread computes the min value, and a third thread computes the average. The program then prints out the average, minimum, and maximum value, which is 44.29, 2, and 98.

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!"**

```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;

}
```

```
treydenwilson@treydenwilson-VirtualBox:~$ gedit
treydenwilson@treydenwilson-VirtualBox:~$ gcc -g Q3Lab4.c -o Q3Lab4
treydenwilson@treydenwilson-VirtualBox:~$ ./Q3Lab4
treydenwilson@treydenwilson-VirtualBox:~$ cat outputLab4.txt
This is a test for opening, writing, and closing a file!treydenwilson@treydenwi
lson-VirtualBox:~$
```

The following code opens or creates the file outputLab4.txt which it appends the phrase “This is a test for opening, writing, and closing a file!”. In addition, the code take precautions and exits upon failures. Thus, it satisfies the requirements for question 3.

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

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

#define ROWS 3
#define COLS 3

int matrix_a[ROWS][COLS];
int matrix_b[ROWS][COLS];
int matrix_c[ROWS][COLS]; // to store the result of the operations
int operation_type; // 0 = addition, 1 = subtraction, 2 = multiplication

void *matrix_operation(void *arg) {
    int row = *(int *) arg;
    for (int j = 0; j < COLS; j++) {
        if (operation_type == 0) {
            matrix_c[row][j] = matrix_a[row][j] + matrix_b[row][j];
        } else if (operation_type == 1) {
            matrix_c[row][j] = matrix_a[row][j] - matrix_b[row][j];
        } else if (operation_type == 2) {
            int sum = 0;
            for (int k = 0; k < ROWS; k++) {
                sum += matrix_a[row][k] * matrix_b[k][j];
            }
            matrix_c[row][j] = sum;
        }
    }
    pthread_exit(NULL);
}

void print_matrix(int matrix[][COLS]) {
    for (int i = 0; i < ROWS; i++) {
        for (int j = 0; j < COLS; j++) {
            printf("%d ", matrix[i][j]);
        }
        printf("\n");
    }
}

int main() {
    pthread_t threads[ROWS];

    // initialize matrix_a and matrix_b with some values
    for (int i = 0; i < ROWS; i++) {
        for (int j = 0; j < COLS; j++) {
            matrix_a[i][j] = i * COLS + j;
            matrix_b[i][j] = j * ROWS + i;
        }
    }

    // perform addition
    operation_type = 0;
```

```
53      printf("Matrix addition:\n");
54      for (int i = 0; i < ROWS; i++) {
55          pthread_create(&threads[i], NULL, matrix_operation, (void *) &i);
56      }
57      for (int i = 0; i < ROWS; i++) {
58          pthread_join(threads[i], NULL);
59      }
60      print_matrix(matrix_c);
61
62      // perform subtraction
63      operation_type = 1;
64      printf("Matrix subtraction:\n");
65      for (int i = 0; i < ROWS; i++) {
66          pthread_create(&threads[i], NULL, matrix_operation, (void *) &i);
67      }
68      for (int i = 0; i < ROWS; i++) {
69          pthread_join(threads[i], NULL);
70      }
71      print_matrix(matrix_c);
72
73      // perform multiplication
74      operation_type = 2;
75      printf("Matrix multiplication:\n");
76      for (int i = 0; i < ROWS; i++) {
77          pthread_create(&threads[i], NULL, matrix_operation, (void *) &i);
78      }
79      for (int i = 0; i < ROWS; i++) {
80          pthread_join(threads[i], NULL);
81      }
82      print_matrix(matrix_c);
83
84      return 0;
85  }
86
```

```
treydenwilson@treydenwilson-VirtualBox:~$ gedit
treydenwilson@treydenwilson-VirtualBox:~$ gcc -g Q4Lab4.c -o Q4Lab4
treydenwilson@treydenwilson-VirtualBox:~$ ./Q4Lab4
Matrix addition:
0 0 0
0 0 0
0 0 0
Matrix subtraction:
0 0 0
0 0 0
0 0 0
Matrix multiplication:
0 0 0
0 0 0
0 0 0
treydenwilson@treydenwilson-VirtualBox:~$
```

The following program conducts matrix addition, subtraction and multiplication. In the screenshot above, zero matrices are used to easily identify that the program works since 0+0 = 0, 0-0 = 0, and 0*0 = 0. Furthermore, the matrix used is a 3x3 matrix and threads are implemented to compute.