

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

```
seer@CS470-Ubuntu-Instance: ~/labs/lab04
seer@CS470-Ubuntu-Instance:~/labs/lab04$ touch myfile.txt
seer@CS470-Ubuntu-Instance:~/labs/lab04$ ls
myfile.txt
seer@CS470-Ubuntu-Instance:~/labs/lab04$ ls -l
total 4
-rw-rw-r-- 1 seer seer 40 Feb 21 18:43 myfile.txt
seer@CS470-Ubuntu-Instance:~/labs/lab04$ ln myfile.txt myhardlink.txt
seer@CS470-Ubuntu-Instance:~/labs/lab04$ ln -s myfile.txt mysoftlink.txt
seer@CS470-Ubuntu-Instance:~/labs/lab04$ ls -l
total 8
-rw-rw-r-- 2 seer seer 40 Feb 21 18:43 myfile.txt
-rw-rw-r-- 2 seer seer 40 Feb 21 18:43 myhardlink.txt
lrwxrwxrwx 1 seer seer 10 Feb 21 18:44 mysoftlink.txt -> myfile.txt
seer@CS470-Ubuntu-Instance:~/labs/lab04$
```

2. Create a multithreaded program that computes different statistical values for a set of numbers.

```
seer@CS470-Ubuntu-Instance: ~/labs/lab04
seer@CS470-Ubuntu-Instance:~/labs/lab04$ gcc lab04.c -o lab04
seer@CS470-Ubuntu-Instance:~/labs/lab04$ ./lab04
The minimum value within the array is 2
The maximum value within the array is 98
seer@CS470-Ubuntu-Instance:~/labs/lab04$
```

```
1 4.c x main
2 #include <pthread.h>
3 #include <stdio.h>
4 #include <stdlib.h>
5
6 int numbers[] = {[0]=2, [1]=20, [2]=25, [3]=5, [4]=70, [5]=90, [6]=98};
7 int numElements = sizeof(numbers) / sizeof(int);
8 int numThreads = 2;
9
10 int minIndex = 0;
11 int maxIndex = 0;
12
13 void *findMinIndex(void *arg) {
14     for(int i = 0; i < numElements; i++) {
15         if(numbers[minIndex] > numbers[i]) minIndex = i;
16     }
17     pthread_exit(retval; NULL);
18 }
19
20 void *findMaxIndex(void *arg) {
21     for(int i = 0; i < numElements; i++) {
22         if(numbers[maxIndex] < numbers[i]) maxIndex = i;
23     }
24     pthread_exit(retval; NULL);
25 }
```

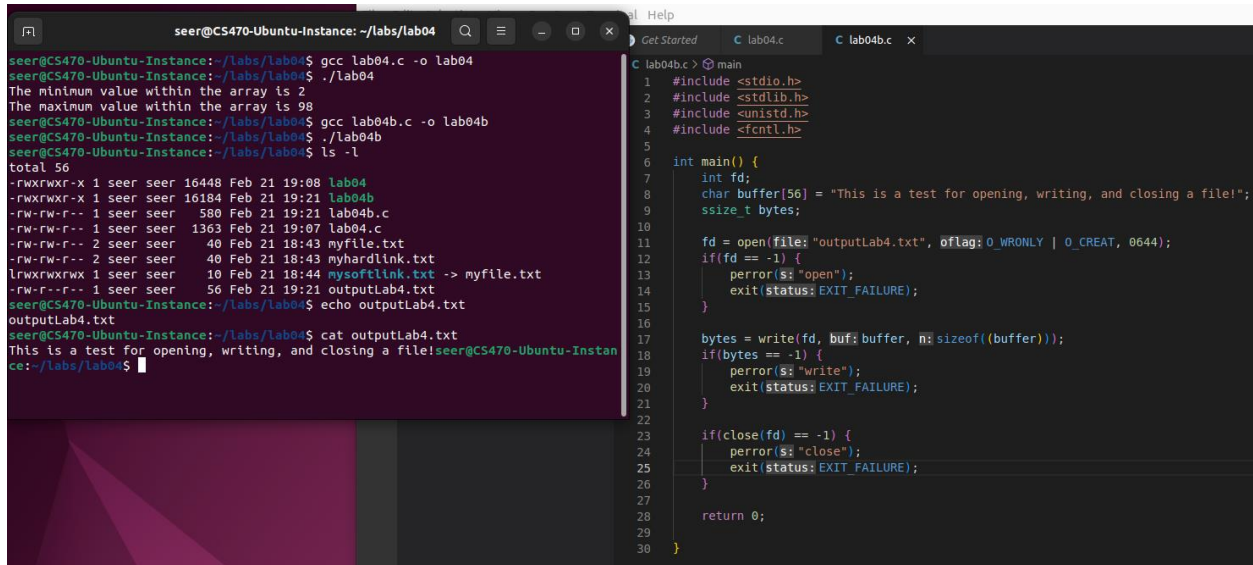
```

26 int main() {
27
28     pthread_t threads[numThreads];
29     int returnCode;
30
31     returnCode = pthread_create(&threads[0], attr: NULL, start_routine: findMinIndex, arg: NULL);
32     if(returnCode) {
33         printf(format: "Error: Unable to create thread.\n");
34         exit(status: -1);
35     }
36
37     returnCode = pthread_create(&threads[1], attr: NULL, start_routine: findMaxIndex, arg: NULL);
38     if(returnCode) {
39         printf(format: "Error: Unable to create thread.\n");
40         exit(status: -1);
41     }
42
43     for(int i = 0; i < numThreads; i++) {
44         returnCode = pthread_join(th: threads[i], thread_return: NULL);
45         if(returnCode) {
46             printf(format: "Error: Unable to join threads.\n");
47             exit(status: -1);
48         }
49     }
50
51     printf(format: "The minimum value within the array is %d\n", numbers[minIndex]);
52     printf(format: "The maximum value within the array is %d\n", numbers[maxIndex]);
53     pthread_exit(retval: NULL);
54     return 0;
55 }

```

The above code utilizes multithreading to find the indices of the minimum and maximum values within the array, spread over two loops. The minimum is found by comparing the value within the index with each other value, and if a smaller value is found, the minimum index is replaced. Similarly, the maximum is found using a similar principle, except comparing against larger values. If a larger value exists, then the maxIndex is set to the index where the higher value was found. Next, the threads are created using pthread_create, and any errors are logged if the threads cannot be created. Finally, the threads are joined back together into the main process once both the minimum and maximum values have been found.

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!”



The image shows a terminal window on the left and a C code editor on the right. The terminal window displays the following commands and output:

```
seer@CS470-Ubuntu-Instance: ~/labs/lab04
seer@CS470-Ubuntu-Instance:~/labs/lab04$ gcc lab04.c -o lab04
seer@CS470-Ubuntu-Instance:~/labs/lab04$ ./lab04
The minimum value within the array is 2
The maximum value within the array is 98
seer@CS470-Ubuntu-Instance:~/labs/lab04$ gcc lab04b.c -o lab04b
seer@CS470-Ubuntu-Instance:~/labs/lab04$ ./lab04b
seer@CS470-Ubuntu-Instance:~/labs/lab04$ ls -l
total 56
-rwxrwxr-x 1 seer seer 16448 Feb 21 19:08 lab04
-rwxrwxr-x 1 seer seer 16184 Feb 21 19:21 lab04b
-rw-rw-r-- 1 seer seer 580 Feb 21 19:21 lab04b.c
-rw-rw-r-- 1 seer seer 1363 Feb 21 19:07 lab04.c
-rw-rw-r-- 2 seer seer 40 Feb 21 18:43 myfile.txt
-rw-rw-r-- 2 seer seer 40 Feb 21 18:43 myhardlink.txt
lrwxrwxrwx 1 seer seer 10 Feb 21 18:44 mysoftlink.txt -> myfile.txt
-rw-r--r-- 1 seer seer 56 Feb 21 19:21 outputLab4.txt
seer@CS470-Ubuntu-Instance:~/labs/lab04$ echo outputLab4.txt
outputLab4.txt
seer@CS470-Ubuntu-Instance:~/labs/lab04$ cat outputLab4.txt
This is a test for opening, writing, and closing a file!seer@CS470-Ubuntu-Instan
ce:~/labs/lab04$
```

The C code editor shows the following code:

```
C lab04b.c > main
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4 #include <fcntl.h>
5
6 int main() {
7     int fd;
8     char buffer[56] = "This is a test for opening, writing, and closing a file!";
9     ssize_t bytes;
10
11     fd = open(file: "outputLab4.txt", oflag: O_WRONLY | O_CREAT, 0644);
12     if(fd == -1) {
13         perror(s: "open");
14         exit(status: EXIT_FAILURE);
15     }
16
17     bytes = write(fd, buf: buffer, n: sizeof(buffer));
18     if(bytes == -1) {
19         perror(s: "write");
20         exit(status: EXIT_FAILURE);
21     }
22
23     if(close(fd) == -1) {
24         perror(s: "close");
25         exit(status: EXIT_FAILURE);
26     }
27
28     return 0;
29
30 }
```

This is a singularly threaded process that creates a file called “outputLab4.txt” in the local folder, writes to it, and finally closes it. The variable “fd” stands for file data, or file descriptor, which contains the current status of the file. “bytes” stores the the number of bytes written to the file, and “buffer” stores a 56 character buffer that contains the text to be written into the file. Each step has a verification check to ensure that the file has been properly written to the operating system, including the closing of the program.

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

```
seer@CS470-Ubuntu-Instance:~/labs/lab04$ gcc lab04c.c -o lab04c
seer@CS470-Ubuntu-Instance:~/labs/lab04$ ./lab04c
Matrix sum:
6, 6, 4, 11,
8, 5, 11, 12,
5, -8, 1, 7,
10, 8, 13, 16,
Matrix difference:
4, 0, 0, 3,
4, -3, -1, 6,
-9, -2, 5, 9,
2, 2, 5, -4,
Matrix product:
5, 9, 4, 28,
12, 4, 30, 27,
-14, 15, -6, -8,
24, 15, 36, 60,
seer@CS470-Ubuntu-Instance:~/labs/lab04$
```

```

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

int matrixA[4][4] = {
    [0]={ [0]=5, [1]=3, [2]=2, [3]=7},
    [1]={ [0]=6, [1]=1, [2]=5, [3]=9},
    [2]={ [0]=-2, [1]=-5, [2]=3, [3]=8},
    [3]={ [0]=6, [1]=5, [2]=9, [3]=6}
};

int matrixB[4][4] = {
    [0]={ [0]=1, [1]=3, [2]=2, [3]=4},
    [1]={ [0]=2, [1]=4, [2]=6, [3]=3},
    [2]={ [0]=7, [1]=-3, [2]=-2, [3]=-1},
    [3]={ [0]=4, [1]=3, [2]=4, [3]=10}
};

int matrixSum[4][4], matrixDiff[4][4], matrixProd[4][4];
int numThreads = 3;

void *matrixAddition(void *arg) {
    for(int i = 0; i < 4; i++) {
        for(int j = 0; j < 4; j++) {
            matrixSum[i][j] = matrixA[i][j] + matrixB[i][j];
        }
    }
    printf(format: "Matrix sum:\n");
    for(int i = 0; i < 4; i++) {
        for(int j = 0; j < 4; j++) {
            printf(format: "%d, ", matrixSum[i][j]);
        }
        printf(format: "\n");
    }
    pthread_exit(retval: NULL);
}

```

```
void *matrixSubtract(void *arg) {
    for(int i = 0; i < 4; i++) {
        for(int j = 0; j < 4; j++) {
            matrixDiff[i][j] = matrixA[i][j] - matrixB[i][j];
        }
    }
    printf(format: "Matrix difference:\n");
    for(int i = 0; i < 4; i++) {
        for(int j = 0; j < 4; j++) {
            printf(format: "%d, ", matrixDiff[i][j]);
        }
        printf(format: "\n");
    }
    pthread_exit(retval: NULL);
}

void *matrixMultiply(void *arg) {
    for(int i = 0; i < 4; i++) {
        for(int j = 0; j < 4; j++) {
            matrixProd[i][j] = matrixA[i][j] * matrixB[i][j];
        }
    }
    printf(format: "Matrix product:\n");
    for(int i = 0; i < 4; i++) {
        for(int j = 0; j < 4; j++) {
            printf(format: "%d, ", matrixProd[i][j]);
        }
        printf(format: "\n");
    }
    pthread_exit(retval: NULL);
}
```

```

int main() {

    pthread_t threads[numThreads];
    int returnCode;

    returnCode = pthread_create(&threads[0], attr: NULL, start_routine: matrixAddition, arg: NULL);
    if(returnCode) {
        printf(format: "Error: Unable to create thread.\n");
        exit(status: -1);
    }

    returnCode = pthread_create(&threads[1], attr: NULL, start_routine: matrixSubtract, arg: NULL);
    if(returnCode) {
        printf(format: "Error: Unable to create thread.\n");
        exit(status: -1);
    }

    returnCode = pthread_create(&threads[2], attr: NULL, start_routine: matrixMultiply, arg: NULL);
    if(returnCode) {
        printf(format: "Error: Unable to create thread.\n");
        exit(status: -1);
    }

    for(int i = 0; i < numThreads; i++) {
        returnCode = pthread_join(th: threads[i], thread_return: NULL);
        if(returnCode) {
            printf(format: "Error: Unable to join threads.\n");
            exit(status: -1);
        }
    }

    pthread_exit(retval: NULL);
    return 0;
}

```

This is a multi-threaded program designed to perform addition, subtraction, and multiplication separately on two matrices. The two matrices, MatrixA and MatrixB, are created with a set of arbitrary values. The functions “matrixAddition”, “matrixSubtract”, and “matrixMultiply” are multithreading-compatible functions that each respectively add, subtract, and multiply two matrices with each other in $O(n^2)$ time. In the main function, the three threads are created, and are verified that they have been created, with each thread tasked with accomplishing a specific function. Finally, after all threads have completed their jobs, the threads are rejoined into the main thread, and the program is exited safely.