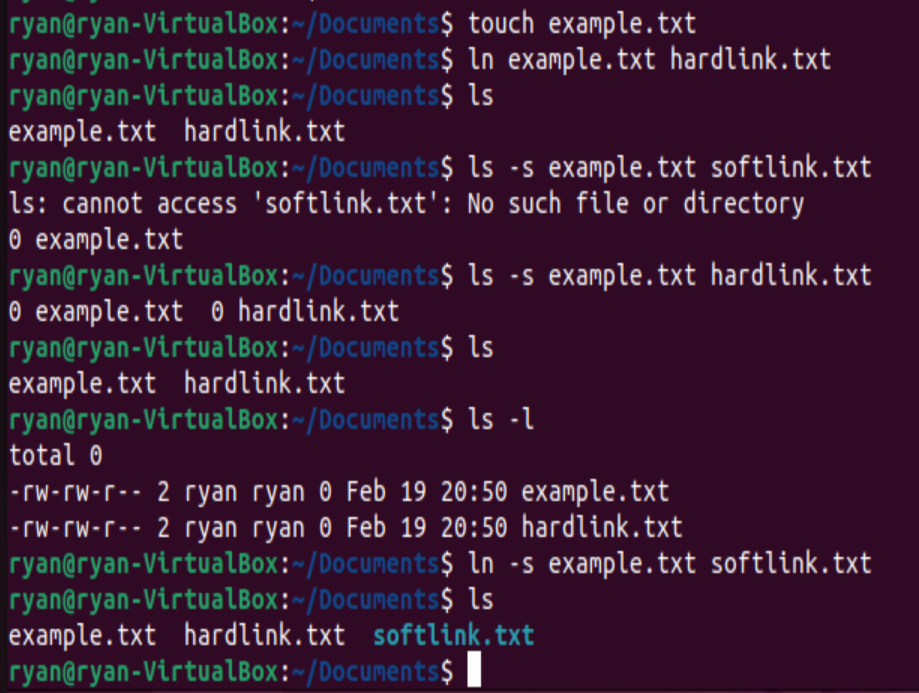
1.

Hard links are made using ln. Softlinks are ln -s.



2. This code creates two threads to run two processes. Then when it closes both threads, it joins them and reports the results.

Code:

#include <pthread.h>

#include <stdio.h>

#include <stdlib.h>

#define NUM\_THREADS 2

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

int num\_count = sizeof(numbers) / sizeof(int);

int max, min;

void calc\_max(voidarg) {

max = numbers[0];

for (int i = 1; i < num\_count; i++) {

if (numbers[i] > max) {

max = numbers[i];

}

}

pthread\_exit(NULL);

}

void calc\_min(voidarg) {

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\_max, NULL);

if (rc) {

printf("Error: Unable to create thread.\n");

exit(-1);

}

rc = pthread\_create(&threads[1], 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);

}

}

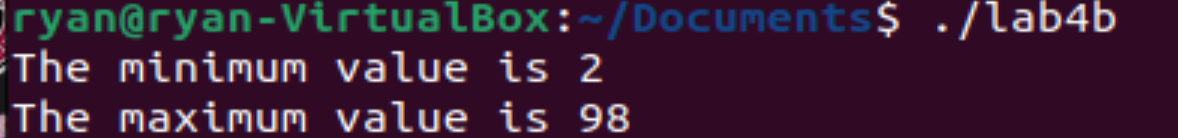
printf("The minimum value is %d\n", min);

printf("The maximum value is %d\n", max);

pthread\_exit(NULL);

}

Output:



3. This code will open the file outputchange.txt, write to it, then close it.

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("outputchange.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");

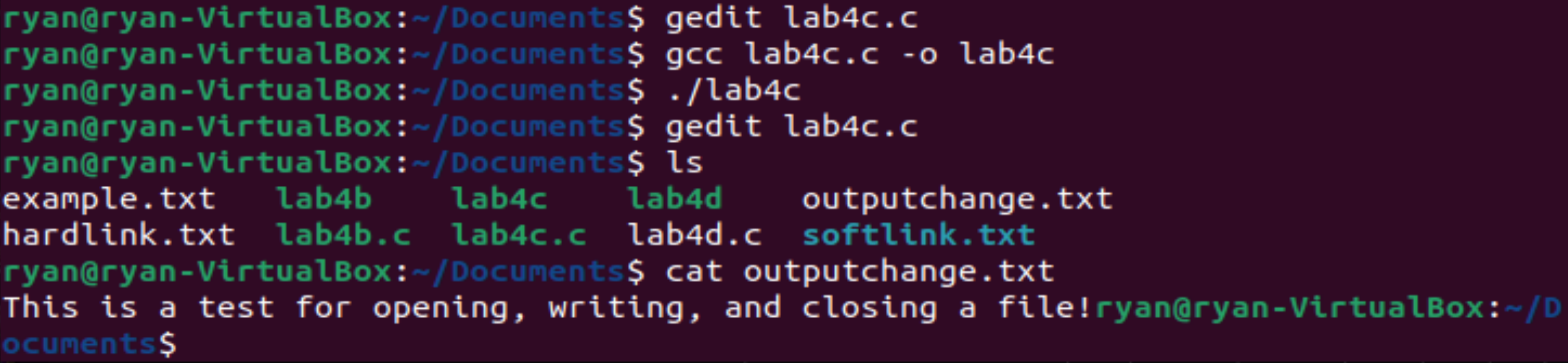
exit(EXIT\_FAILURE);

}

return 0;

}

Output:



4. The following code makes three threads, which execute three matrix operations. After exiting, they are joined. The results are then reported.

Code:

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#define ROWS 3

#define COLS 3

// define matrices

int matrixA[ROWS][COLS];

int matrixB[ROWS][COLS];

int matrixSum[ROWS][COLS];

int matrixDiff[ROWS][COLS];

int matrixProd[ROWS][COLS];

// define struct for passing arguments to threads

typedef struct {

int row;

int col;

} thread\_args;

// thread function for matrix addition

void \*add\_matrix(void \*args) {

thread\_args \*targs = (thread\_args \*) args;

int row = targs->row;

int col = targs->col;

matrixSum[row][col] = matrixA[row][col] + matrixB[row][col];

pthread\_exit(NULL);

}

// thread function for matrix subtraction

void \*sub\_matrix(void \*args) {

thread\_args \*targs = (thread\_args \*) args;

int row = targs->row;

int col = targs->col;

matrixDiff[row][col] = matrixA[row][col] - matrixB[row][col];

pthread\_exit(NULL);

}

// thread function for matrix multiplication

void \*mul\_matrix(void \*args) {

thread\_args \*targs = (thread\_args \*) args;

int row = targs->row;

int col = targs->col;

matrixProd[row][col] = 0;

for (int k = 0; k < COLS; k++) {

matrixProd[row][col] += matrixA[row][k] \* matrixB[k][col];

}

pthread\_exit(NULL);

}

int main() {

pthread\_t threads[ROWS][COLS];

thread\_args targs[ROWS][COLS];

// initialize matrices with random values

printf("Matrix A:\n");

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

for (int j = 0; j < COLS; j++) {

matrixA[i][j] = rand() % 10;

printf("%d ", matrixA[i][j]);

}

printf("\n");

}

printf("\n");

printf("Matrix B:\n");

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

for (int j = 0; j < COLS; j++) {

matrixB[i][j] = rand() % 10;

printf("%d ", matrixB[i][j]);

}

printf("\n");

}

printf("\n");

// create threads for matrix addition

printf("Matrix A + Matrix B:\n");

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

for (int j = 0; j < COLS; j++) {

targs[i][j].row = i;

targs[i][j].col = j;

pthread\_create(&threads[i][j], NULL, add\_matrix, (void \*) &targs[i][j]);

}

}

// join threads for matrix addition

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

for (int j = 0; j < COLS; j++) {

pthread\_join(threads[i][j], NULL);

printf("%d ", matrixSum[i][j]);

}

printf("\n");

}

printf("\n");

// create threads for matrix subtraction

printf("Matrix A - Matrix B:\n");

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

for (int j = 0; j < COLS; j++) {

targs[i][j].row = i;

targs[i][j].col = j;

pthread\_create(&threads[i][j], NULL, sub\_matrix, (void \*) &targs[i][j]);

}

}

// join threads for matrix subtraction

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

for (int j = 0; j < COLS; j++) {

pthread\_join(threads[i][j], NULL);

printf("%d ", matrixDiff[i][j]);

}

printf("\n");

}

printf("\n");

// create threads for matrix multiplication

printf("Matrix A \* Matrix B:\n");

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

for (int j = 0; j < COLS; j++) {

targs[i][j].row = i;

targs[i][j].col = j;

pthread\_create(&threads[i][j], NULL, mul\_matrix, (void \*) &targs[i][j]);

}

}

// join threads for matrix multiplication

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

for (int j = 0; j < COLS; j++) {

pthread\_join(threads[i][j], NULL);

printf("%d ", matrixProd[i][j]);

}

printf("\n");

}

printf("\n");

return 0; }

Output:

Text

Description automatically generated