WASN'T ABLE TO SCREENSHOT IN LINUX

Create hard and soft link using ln and ln-s commands

1. Hardlink: $ ln file.txt newlink.txt
   Softlink: $ ln -s file.txt newlink.txt

2. Calculates the minimum and maximium values of a list of integers using two independent worker threads.

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

#define THREADS 2

int *numbers;
int num_count;

int min_val = 0, max_val = 0;

void *compute_min(void *arg) {
    int i;
    for (i = 0; i < num_count; i++) {
        if (numbers[i] < min_val)
            min_val = numbers[i];
    }
    pthread_exit(NULL);
}

void *compute_max(void *arg) {
    int i;
    for (i = 0; i < num_count; i++) {
        if (numbers[i] > max_val)
            max_val = numbers[i];
    }
    pthread_exit(NULL);
}

int main(int argc, char *argv[]) {
    pthread_t threads[THREADS];
    int i;

    if (argc < 2) {
        printf("Usage: %s num1 num2 num3 ...\n", argv[0]);
        return 1;
    }

    num_count = argc - 1;
    numbers = (int *) malloc(num_count * sizeof(int));
    for (i = 0; i < num_count; i++) {
        numbers[i] = atoi(argv[i+1]);
    }

    pthread_create(&threads[0], NULL, compute_min, NULL);
    pthread_create(&threads[1], NULL, compute_max, NULL);

    for (i = 0; i < THREADS; i++) {
        pthread_join(threads[i], NULL);
    }

    printf("The min is %d.\n", min_val);
    printf("The max is %d.\n", max_val);

    free(numbers);
```

```
    return 0;
}

```

3. Here's an example C program that opens a file named "outputLab4.txt" for writing and appends the phrase "This is a test for opening, writing, and closing a file!" to it:

```c
#include <stdio.h>

int main() {
    FILE *fp;

    fp = fopen("outputLab4.txt", "a");
    if (fp == NULL) {
        printf("Error opening file!\n");
        return 1;
    }

    fprintf(fp, "This is a test for opening, writing, and closing a file!\n");

    fclose(fp);
    return 0;
}

```

4. here is the code for matrix addition, subtraction, and multiplication using multithreading in C programming language:

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

#define MAX_SIZE 100

int A[MAX_SIZE][MAX_SIZE], B[MAX_SIZE][MAX_SIZE], C[MAX_SIZE][MAX_SIZE];
int m, n, p, q;

void* matrix_addition(void* arg) {
    int thread_id = *((int*) arg);
    int chunk_size = m / thread_id;
    int start = (thread_id - 1) * chunk_size;
    int end = (thread_id == n) ? m : start + chunk_size;

    for (int i = start; i < end; i++) {
        for (int j = 0; j < p; j++) {
            C[i][j] = A[i][j] + B[i][j];
        }
    }
}

void* matrix_subtraction(void* arg) {
    int thread_id = *((int*) arg);
    int chunk_size = m / thread_id;
    int start = (thread_id - 1) * chunk_size;
    int end = (thread_id == n) ? m : start + chunk_size;

    for (int i = start; i < end; i++) {
        for (int j = 0; j < p; j++) {
            C[i][j] = A[i][j] - B[i][j];
        }
    }
}

void* matrix_multiplication(void* arg) {
    int thread_id = *((int*) arg);
    int chunk_size = m / thread_id;
    int start = (thread_id - 1) * chunk_size;
    int end = (thread_id == n) ? m : start + chunk_size;

    for (int i = start; i < end; i++) {
        for (int j = 0; j < q; j++) {
            C[i][j] = 0;
            for (int k = 0; k < n; k++) {
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }
}

void input_matrices() {
    printf("Enter the number of rows and columns of Matrix A:\n");
    scanf("%d %d", &m, &n);

    printf("Enter the elements of Matrix A:\n");
```

```c
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &A[i][j]);
        }
    }

    printf("Enter the number of rows and columns of Matrix B:\n");
    scanf("%d %d", &p, &q);

    printf("Enter the elements of Matrix B:\n");
    for (int i = 0; i < p; i++) {
        for (int j = 0; j < q; j++) {
            scanf("%d", &B[i][j]);
        }
    }
}

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

int main() {
    int choice;
    pthread_t threads[MAX_SIZE];

    input_matrices();

    printf("Select an operation:\n");
    printf("1. Matrix Addition\n");
    printf("2. Matrix Subtraction\n");
    printf("3. Matrix Multiplication\n");
    scanf("%d", &choice);
    switch(choice) {
        case 1: {
            if (m != p || n != q) {
                printf("Matrices A and B cannot be added!\n");
                break;
            }

            printf("Performing Matrix Addition using %d threads...\n", MAX_SIZE);
            for (int i = 1; i <= MAX_SIZE; i++) {
                int* arg = (int*) malloc(sizeof(*arg));
                *arg = i;
                pthread_create(&threads[i], NULL, matrix_addition, arg);
            }

            for (int i = 1; i <= MAX_SIZE; i++) {
                pthread_join(threads[i], NULL);
            }

```

```
            printf("Resultant Matrix C:\n");
            print_matrix(C, m, n);
            break;
        }
        case 2: {
            if (m != p || n != q) {
                printf("Matrices A and B cannot be subtracted!\n");
                break;
            }

            printf("Performing Matrix Subtraction using %d threads...\n", MAX_SIZE);
            for (int i = 1; i <= MAX_SIZE; i++) {
                int* arg = (int*) malloc(sizeof(*arg));
                *arg = i;
                pthread_create(&threads[i], NULL, matrix_subtraction, arg);
            }

            for (int i = 1; i <= MAX_SIZE; i++) {
                pthread_join(threads[i], NULL);
            }

            printf("Resultant Matrix C:\n");
            print_matrix(C, m, n);
            break;
        }
        case 3: {
            if (n != p) {
                printf("Matrices A and B cannot be multiplied!\n");
                break;
            }

            printf("Performing Matrix Multiplication using %d threads...\n", MAX_SIZE);
            for (int i = 1; i <= MAX_SIZE; i++) {
                int* arg = (int*) malloc(sizeof(*arg));
                *arg = i;
                pthread_create(&threads[i], NULL, matrix_multiplication, arg);
            }

            for (int i = 1; i <= MAX_SIZE; i++) {
                pthread_join(threads[i], NULL);
            }

            printf("Resultant Matrix C:\n");
            print_matrix(C, m, q);
            break;
        }
        default: {
            printf("Invalid choice!\n");
            break;
        }
    }

    return 0;
}
```