

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 maximum values of a list of integers using two independent worker threads.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <pthread.h>
4
5  #define THREADS 2
6
7  int *numbers;
8  int num_count;
9
10 int min_val = 0, max_val = 0;
11
12 void *compute_min(void *arg) {
13     int i;
14     for (i = 0; i < num_count; i++) {
15         if (numbers[i] < min_val)
16             min_val = numbers[i];
17     }
18     pthread_exit(NULL);
19 }
20
21 void *compute_max(void *arg) {
22     int i;
23     for (i = 0; i < num_count; i++) {
24         if (numbers[i] > max_val)
25             max_val = numbers[i];
26     }
27     pthread_exit(NULL);
28 }
29
30 int main(int argc, char *argv[]) {
31     pthread_t threads[THREADS];
32     int i;
33
34     if (argc < 2) {
35         printf("Usage: %s num1 num2 num3 ...\n", argv[0]);
36         return 1;
37     }
38
39     num_count = argc - 1;
40     numbers = (int *) malloc(num_count * sizeof(int));
41     for (i = 0; i < num_count; i++) {
42         numbers[i] = atoi(argv[i+1]);
43     }
44
45     pthread_create(&threads[0], NULL, compute_min, NULL);
46     pthread_create(&threads[1], NULL, compute_max, NULL);
47
48     for (i = 0; i < THREADS; i++) {
49         pthread_join(threads[i], NULL);
50     }
51
52     printf("The min is %d.\n", min_val);
53     printf("The max is %d.\n", max_val);
54
55     free(numbers);
```

```
56     return 0;
57 }
58
```

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:

```
1  #include <stdio.h>
2
3  int main() {
4      FILE *fp;
5
6      fp = fopen("outputLab4.txt", "a");
7      if (fp == NULL) {
8          printf("Error opening file!\n");
9          return 1;
10     }
11
12     fprintf(fp, "This is a test for opening, writing, and closing a file!\n");
13
14     fclose(fp);
15     return 0;
16 }
17
```

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

```

1  #include <stdio.h>
2  #include <pthread.h>
3
4  #define MAX_SIZE 100
5
6  int A[MAX_SIZE][MAX_SIZE], B[MAX_SIZE][MAX_SIZE], C[MAX_SIZE][MAX_SIZE];
7  int m, n, p, q;
8
9  void* matrix_addition(void* arg) {
10     int thread_id = *((int*) arg);
11     int chunk_size = m / thread_id;
12     int start = (thread_id - 1) * chunk_size;
13     int end = (thread_id == n) ? m : start + chunk_size;
14
15     for (int i = start; i < end; i++) {
16         for (int j = 0; j < p; j++) {
17             C[i][j] = A[i][j] + B[i][j];
18         }
19     }
20 }
21
22 void* matrix_subtraction(void* arg) {
23     int thread_id = *((int*) arg);
24     int chunk_size = m / thread_id;
25     int start = (thread_id - 1) * chunk_size;
26     int end = (thread_id == n) ? m : start + chunk_size;
27
28     for (int i = start; i < end; i++) {
29         for (int j = 0; j < p; j++) {
30             C[i][j] = A[i][j] - B[i][j];
31         }
32     }
33 }
34
35 void* matrix_multiplication(void* arg) {
36     int thread_id = *((int*) arg);
37     int chunk_size = m / thread_id;
38     int start = (thread_id - 1) * chunk_size;
39     int end = (thread_id == n) ? m : start + chunk_size;
40
41     for (int i = start; i < end; i++) {
42         for (int j = 0; j < q; j++) {
43             C[i][j] = 0;
44             for (int k = 0; k < n; k++) {
45                 C[i][j] += A[i][k] * B[k][j];
46             }
47         }
48     }
49 }
50
51 void input_matrices() {
52     printf("Enter the number of rows and columns of Matrix A:\n");
53     scanf("%d %d", &m, &n);
54
55     printf("Enter the elements of Matrix A:\n");

```

```

56     for (int i = 0; i < m; i++) {
57         for (int j = 0; j < n; j++) {
58             scanf("%d", &A[i][j]);
59         }
60     }
61
62     printf("Enter the number of rows and columns of Matrix B:\n");
63     scanf("%d %d", &p, &q);
64
65     printf("Enter the elements of Matrix B:\n");
66     for (int i = 0; i < p; i++) {
67         for (int j = 0; j < q; j++) {
68             scanf("%d", &B[i][j]);
69         }
70     }
71 }
72
73 void print_matrix(int matrix[MAX_SIZE][MAX_SIZE], int rows, int cols) {
74     for (int i = 0; i < rows; i++) {
75         for (int j = 0; j < cols; j++) {
76             printf("%d ", matrix[i][j]);
77         }
78         printf("\n");
79     }
80 }
81
82 int main() {
83     int choice;
84     pthread_t threads[MAX_SIZE];
85
86     input_matrices();
87
88     printf("Select an operation:\n");
89     printf("1. Matrix Addition\n");
90     printf("2. Matrix Subtraction\n");
91     printf("3. Matrix Multiplication\n");
92     scanf("%d", &choice);
93     switch(choice) {
94         case 1: {
95             if (m != p || n != q) {
96                 printf("Matrices A and B cannot be added!\n");
97                 break;
98             }
99
100            printf("Performing Matrix Addition using %d threads...\n", MAX_SIZE);
101            for (int i = 1; i <= MAX_SIZE; i++) {
102                int* arg = (int*) malloc(sizeof(*arg));
103                *arg = i;
104                pthread_create(&threads[i], NULL, matrix_addition, arg);
105            }
106
107            for (int i = 1; i <= MAX_SIZE; i++) {
108                pthread_join(threads[i], NULL);
109            }
110

```

```

111         printf("Resultant Matrix C:\n");
112         print_matrix(C, m, n);
113         break;
114     }
115     case 2: {
116         if (m != p || n != q) {
117             printf("Matrices A and B cannot be subtracted!\n");
118             break;
119         }
120
121         printf("Performing Matrix Subtraction using %d threads...\n", MAX_SIZE);
122         for (int i = 1; i <= MAX_SIZE; i++) {
123             int* arg = (int*) malloc(sizeof(*arg));
124             *arg = i;
125             pthread_create(&threads[i], NULL, matrix_subtraction, arg);
126         }
127
128         for (int i = 1; i <= MAX_SIZE; i++) {
129             pthread_join(threads[i], NULL);
130         }
131
132         printf("Resultant Matrix C:\n");
133         print_matrix(C, m, n);
134         break;
135     }
136     case 3: {
137         if (n != p) {
138             printf("Matrices A and B cannot be multiplied!\n");
139             break;
140         }
141
142         printf("Performing Matrix Multiplication using %d threads...\n", MAX_SIZE);
143         for (int i = 1; i <= MAX_SIZE; i++) {
144             int* arg = (int*) malloc(sizeof(*arg));
145             *arg = i;
146             pthread_create(&threads[i], NULL, matrix_multiplication, arg);
147         }
148
149         for (int i = 1; i <= MAX_SIZE; i++) {
150             pthread_join(threads[i], NULL);
151         }
152
153         printf("Resultant Matrix C:\n");
154         print_matrix(C, m, q);
155         break;
156     }
157     default: {
158         printf("Invalid choice!\n");
159         break;
160     }
161 }
162
163 return 0;
164 }

```