122CS0067

Assignment-VI

Name: **Amiya Chowdhury** Roll: **122CS0067** Date: **21/10/2024**

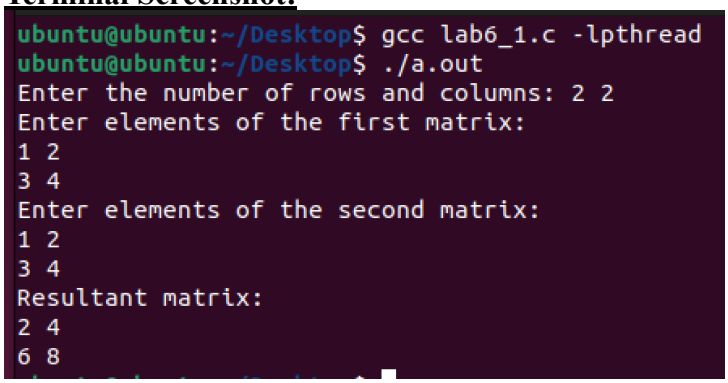
1. Input and add two (mxn) matrices. Main thread creates m child threads. Thread1 computes the 1st row, Thread2 computes the 2nd row, …, Thread m computes the mth row elements of the result matrix.

lab6_1.c

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

struct arg_struct {
    int row;            // Row index for the row to be processed
    int **mat1;         // Pointer to the first matrix
    int **mat2;         // Pointer to the second matrix
    int **result;       // Pointer to the result matrix
    int cols;           // Number of columns (n)
};

void *add_row(void *arguments) {
    struct arg_struct *args = (struct arg_struct *)arguments;

    // Process the entire row by adding corresponding elements
    for (int col = 0; col < args->cols; col++) {
        args->result[args->row][col] = args->mat1[args->row][col] + args->mat2[args->row][col];
    }

    pthread_exit(NULL);
}

int main() {
    int m, n;
    printf("Enter the number of rows and columns: ");
    scanf("%d %d", &m, &n);

    // memory for mat1
    int **mat1 = malloc(m * sizeof(int *));
    for (int i = 0; i < m; i++) {
        mat1[i] = malloc(n * sizeof(int));
    }

    // mat2
    int **mat2 = malloc(m * sizeof(int *));
    for (int i = 0; i < m; i++) {
        mat2[i] = malloc(n * sizeof(int));
    }

    // result
    int **result = malloc(m * sizeof(int *));
    for (int i = 0; i < m; i++) {
        result[i] = malloc(n * sizeof(int));
    }

    // Input mat1
    printf("Enter elements of the first matrix:\n");
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &mat1[i][j]);
```

```
        }
    }

    // Input mat2
    printf("Enter elements of the second matrix:\n");
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &mat2[i][j]);
        }
    }

    pthread_t threads[m]; // array of threads
    struct arg_struct args[m]; // array of arguments for the threads

    // Creation threads
    for (int i = 0; i < m; i++) {
        args[i].row = i;
        args[i].mat1 = mat1;
        args[i].mat2 = mat2;
        args[i].result = result;
        args[i].cols = n;
        pthread_create(&threads[i], NULL, add_row, (void *)&args[i]);
    }

    // wait for the threads
    for (int i = 0; i < m; i++) {
        pthread_join(threads[i], NULL);
    }

    // output
    printf("Resultant matrix:\n");
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            printf("%d ", result[i][j]);
        }
        printf("\n");
    }

    // freeing up
    for (int i = 0; i < m; i++) {
        free(mat1[i]);
        free(mat2[i]);
        free(result[i]);
    }
    free(mat1);
    free(mat2);
    free(result);

    return 0;
}
```

Terminal Screenshot:



2. Modify the program to create m*n threads.

lab6_2.c

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

struct arg_struct {
    int row;
    int col;
    int **mat1;
    int **mat2;
    int **result;
};

void *add_element(void *arguments) {
    struct arg_struct *args = (struct arg_struct *)arguments;

    // Add corresponding elements and store in the result matrix
    args->result[args->row][args->col] = args->mat1[args->row][args->col] + args->mat2[args->row][args->col];

    pthread_exit(NULL);
}

int main() {
    int m, n;
    printf("Enter the number of rows and columns: ");
    scanf("%d %d", &m, &n);


    int **mat1 = malloc(m * sizeof(int *));
    for (int i = 0; i < m; i++) {
        mat1[i] = malloc(n * sizeof(int));
    }


    int **mat2 = malloc(m * sizeof(int *));
    for (int i = 0; i < m; i++) {
        mat2[i] = malloc(n * sizeof(int));
    }


    int **result = malloc(m * sizeof(int *));
    for (int i = 0; i < m; i++) {
        result[i] = malloc(n * sizeof(int));
    }

    printf("Enter elements of the first matrix:\n");
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &mat1[i][j]);
        }
    }

    printf("Enter elements of the second matrix:\n");
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &mat2[i][j]);
        }
    }
```

```
    pthread_t threads[m][n];
    struct arg_struct args[m][n];

    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            args[i][j].row = i;
            args[i][j].col = j;
            args[i][j].mat1 = mat1;
            args[i][j].mat2 = mat2;
            args[i][j].result = result;
            pthread_create(&threads[i][j], NULL, add_element, (void *)&args[i][j]);
        }
    }

    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            pthread_join(threads[i][j], NULL);
        }
    }

    printf("Resultant matrix:\n");
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            printf("%d ", result[i][j]);
        }
        printf("\n");
    }

        for (int i = 0; i < m; i++) {
        free(mat1[i]);
        free(mat2[i]);
        free(result[i]);
    }
    free(mat1);
    free(mat2);
    free(result);

    return 0;
}
```

Terminal Screenshot: