Bansilal Ramnath Agarwal Charitable Trust's

Vishwakarma Institute of Technology, Pune-37

(An Autonomous Institute of Savitribai Phule Pune University)

Department of AIDS

Division	A
Roll Number	78
PRN Number	12320056
Name	Mayank Kulkarni

Title: Implement multithreading for Matrix Operations using Pthreads.

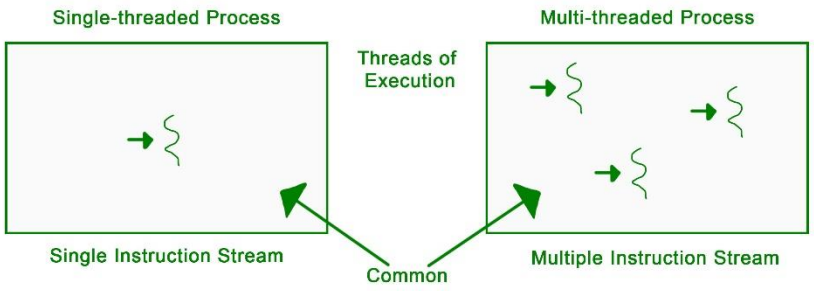
Theory:

What is Multithreading?

Multithreading is a feature in operating systems that allows a program to do several tasks at the same time. Think of it like having multiple hands working together to complete different parts of a job faster. Each "hand" is called a thread, and they help make programs run more efficiently. Multithreading makes your computer work better by using its resources more effectively, leading to quicker and smoother performance for applications like web browsers, games, and many other programs you use every day.

How Does Multithreading Work?

- Multithreading works by allowing a computer's processor to handle multiple tasks at the same time. Even though the processor can only do one thing at a time, it switches between different threads from various programs so quickly that it looks like everything is happening all at once.
- **Processor Handling**: The processor can execute only one instruction at a time, but it switches between different threads so fast that it gives the illusion of simultaneous execution.
- **Thread Synchronization**: Each thread is like a separate task within a program. They share resources and work together smoothly, ensuring programs run efficiently.
- **Efficient Execution**: Threads in a program can run independently or wait for their turn to process, making programs faster and more responsive.
- **Programming Considerations**: Programmers need to be careful about managing threads to avoid problems like conflicts or situations where threads get stuck waiting for each other.



Single Thread and Multi Thread Process

Program:

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

int A[MAX][MAX], B[MAX][MAX], C[MAX][MAX];

typedef struct {
    int row;
    int col;
} ThreadData;

void* matrixAddition(void* arg) {
    ThreadData* data = (ThreadData*)arg;
    int i = data->row;
    int j = data->col;

    C[i][j] = A[i][j] + B[i][j];
    pthread_exit(0);
}

void* matrixMultiplication(void* arg) {
    ThreadData* data = (ThreadData*)arg;
    int i = data->row;
    int j = data->col;

    C[i][j] = 0;
    // Perform multiplication for each element
    for (int k = 0; k < MAX; k++) {
        C[i][j] += A[i][k] * B[k][j];
    }
    pthread_exit(0);
}

void inputMatrices() {
    printf("Enter elements of matrix A (%d x %d):\n", MAX, MAX);
    for (int i = 0; i < MAX; i++) {
        for (int j = 0; j < MAX; j++) {
            printf("A[%d][%d]: ", i, j);
            scanf("%d", &A[i][j]);
        }
    }

    printf("\nEnter elements of matrix B (%d x %d):\n", MAX, MAX);
    for (int i = 0; i < MAX; i++) {
        for (int j = 0; j < MAX; j++) {
            printf("B[%d][%d]: ", i, j);
            scanf("%d", &B[i][j]);
```

```
        }
    }
}

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

int main() {
    pthread_t threads[MAX * MAX];
    ThreadData thread_data[MAX * MAX];
    int thread_count = 0;

    inputMatrices();

    printf("\nPerforming Matrix Addition:\n");
    for (int i = 0; i < MAX; i++) {
        for (int j = 0; j < MAX; j++) {
            thread_data[thread_count].row = i;
            thread_data[thread_count].col = j;
            pthread_create(&threads[thread_count], NULL, matrixAddition,
&thread_data[thread_count]);
            thread_count++;
        }
    }

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

    printf("Result of Matrix Addition:\n");
    printMatrix(C);

    thread_count = 0;

    printf("\nPerforming Matrix Multiplication:\n");
    for (int i = 0; i < MAX; i++) {
        for (int j = 0; j < MAX; j++) {
            thread_data[thread_count].row = i;
            thread_data[thread_count].col = j;
            pthread_create(&threads[thread_count], NULL, matrixMultiplication,
&thread_data[thread_count]);
            thread_count++;
```

```
        }
    }

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

    printf("Result of Matrix Multiplication:\n");
    printMatrix(C);

    return 0;
}
```

Output:

Enter elements of matrix A (4 x 4):

A[0][0]: 1
A[0][1]: 2
A[0][2]: 3
A[0][3]: 4
A[1][0]: 5
A[1][1]: 6
A[1][2]: 7
A[1][3]: 8
A[2][0]: 9
A[2][1]: 10
A[2][2]: 11
A[2][3]: 12
A[3][0]: 13
A[3][1]: 14
A[3][2]: 15
A[3][3]: 16

Enter elements of matrix B (4 x 4):

B[0][0]: 16
B[0][1]: 15
B[0][2]: 14
B[0][3]: 13
B[1][0]: 12

B[1][1]: 11
B[1][2]: 10
B[1][3]: 9
B[2][0]: 8
B[2][1]: 7
B[2][2]: 6
B[2][3]: 5
B[3][0]: 4
B[3][1]: 3
B[3][2]: 2
B[3][3]: 1

Performing Matrix Addition:
Result of Matrix Addition:
17 17 17 17
17 17 17 17
17 17 17 17
17 17 17 17

Performing Matrix Multiplication:
Result of Matrix Multiplication:
80 70 60 50
240 214 188 162
400 358 316 274
560 502 444 386

Conclusion:

Thus, in this practical we have successfully performed and executed the multithreading program showcasing various mathematical and arithmetic operations on the matrix.