// Implement Multithreading for matrix operation using PThreads

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

// Define the dimensions of the matrices

#define M 3

#define N 3

#define P 3

// Matrices

int A[M][N];

int B[N][P];

int C[M][P];

// Structure to pass data to threads

typedef struct {

int row;

int col;

} thread\_data\_t;

// Thread function to perform matrix multiplication for a specific element

void\* multiply(void\* arg) {

thread\_data\_t\* data = (thread\_data\_t\*) arg;

int row = data->row;

int col = data->col;

C[row][col] = 0;

// Perform the matrix multiplication for a single element

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

C[row][col] += A[row][k] \* B[k][col];

}

pthread\_exit(NULL);

}

int main() {

pthread\_t threads[M \* P];

thread\_data\_t thread\_data[M \* P];

// Initialize matrices A and B

printf("Matrix A:\n");

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

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

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

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

}

printf("\n");

}

printf("\nMatrix B:\n");

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

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

B[i][j] = rand() % 10; // Random numbers between 0 and 9

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

}

printf("\n");

}

int thread\_idx = 0;

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

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

thread\_data[thread\_idx].row = i;

thread\_data[thread\_idx].col = j;

pthread\_create(&threads[thread\_idx], NULL, multiply, (void\*)&thread\_data[thread\_idx]);

thread\_idx++;

}

}

// Join all threads

for (int i = 0; i < M \* P; i++) {

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

}

// Display the result matrix C

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

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

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

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

}

printf("\n");

}

return 0;

}