Barriers example

written by Nik Tsonev

A barrier is exactly what it sounds like, a function that waits until all threads have reached it to proceed. At first their use is not so intuitive but they can be used for:

1. Parallel Sorting Algorithms
2. Numerical Solutions
3. Image processing
4. Matrix factorisation

in this case I'll show example 4

we need to compute some matrix and then scaled it, the computation is done in parallel but we must ensure that the matrix has first been computed and then scaled

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

#define MATRIX_SIZE 100
#define NUM_THREADS 4

pthread_mutex_t barrier_mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t barrier_cond = PTHREAD_COND_INITIALIZER;
int count = 0;

void barrier() {
    pthread_mutex_lock(&barrier_mutex);
    count++;

    if (count == NUM_THREADS) {
        count = 0;  // Reset for next use of the barrier
        pthread_cond_broadcast(&barrier_cond);
```

```
    } else {
        while (count != 0) {
            pthread_cond_wait(&barrier_cond,
&barrier_mutex);
    }
    pthread_mutex_unlock(&barrier_mutex);
}



void* matrix_multiply(void* arg) {
    int thread_id = *(int*)arg;



    // Each thread computes a portion of the matrix
    for (int i = thread_id * (MATRIX_SIZE / NUM_THREADS); i
< (thread_id + 1) * (MATRIX_SIZE / NUM_THREADS); i++) {

        for (int j = 0; j < MATRIX_SIZE; j++) {
            // Compute the dot product of row i and column
j
            // (Simulate work)
            sleep(1);  // Simulating work with sleep
        }

    }



    printf("Thread %d has finished its part of the
computation.\n", thread_id);
    barrier(); // ensure all of them wait up until this
point

    // Next phase of processing (e.g., scaling matrix C)
    if (thread_id == 0)
        printf("All threads have reached the barrier.
Proceeding to the next phase.\n");
```

```
    return NULL;
}
```