// Implementation of classical problems ( reader writer) using threads & mutex

#include <pthread.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

// Shared data and synchronization variables

int shared\_data = 0;

int reader\_count = 0;

pthread\_mutex\_t resource\_mutex; // Mutex for exclusive access to shared data

pthread\_mutex\_t reader\_count\_mutex; // Mutex for protecting reader count

// Reader function

void\* reader(void\* arg) {

int reader\_id = \*((int\*)arg);

free(arg);

while (1) {

// Entry section for reader

pthread\_mutex\_lock(&reader\_count\_mutex);

reader\_count++;

if (reader\_count == 1) {

pthread\_mutex\_lock(&resource\_mutex); // First reader locks the resource

}

pthread\_mutex\_unlock(&reader\_count\_mutex);

// Reading section

printf("Reader %d is reading the shared data: %d\n", reader\_id, shared\_data);

usleep(rand() % 500000); // Simulate reading time

// Exit section for reader

pthread\_mutex\_lock(&reader\_count\_mutex);

reader\_count--;

if (reader\_count == 0) {

pthread\_mutex\_unlock(&resource\_mutex); // Last reader unlocks the resource

}

pthread\_mutex\_unlock(&reader\_count\_mutex);

// Random sleep to simulate time between reads

usleep(rand() % 500000);

}

return NULL;

}

// Writer function

void\* writer(void\* arg) {

int writer\_id = \*((int\*)arg);

free(arg);

while (1) {

// Writing section

pthread\_mutex\_lock(&resource\_mutex); // Writer locks the resource

shared\_data++;

printf("Writer %d updated shared data to: %d\n", writer\_id, shared\_data);

usleep(rand() % 600000); // Simulate writing time

pthread\_mutex\_unlock(&resource\_mutex); // Writer releases the resource

// Random sleep to simulate time between writes

usleep(rand() % 1000000);

}

return NULL;

}

int main() {

pthread\_t readers[5], writers[2];

pthread\_mutex\_init(&resource\_mutex, NULL);

pthread\_mutex\_init(&reader\_count\_mutex, NULL);

// Create reader threads

for (int i = 0; i < 5; i++) {

int\* id = malloc(sizeof(int));

\*id = i + 1;

pthread\_create(&readers[i], NULL, reader, id);

}

// Create writer threads

for (int i = 0; i < 2; i++) {

int\* id = malloc(sizeof(int));

\*id = i + 1;

pthread\_create(&writers[i], NULL, writer, id);

}

// Wait for threads to complete (they won't in this infinite loop example)

for (int i = 0; i < 5; i++) {

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

}

for (int i = 0; i < 2; i++) {

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

}

// Destroy mutexes (unreachable in this example but good practice)

pthread\_mutex\_destroy(&resource\_mutex);

pthread\_mutex\_destroy(&reader\_count\_mutex);

return 0;

}