Tutorato-Sistemi-Operativi-2025-main\Esame-[2025-12-05]\magic_square_screener.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#include <string.h>
#include <unistd.h>

/* Costanti definite dalla traccia */
#define QUEUE_INTERMEDIATE_SIZE 10
#define QUEUE_FINAL_SIZE 3
#define POISON_PILL -1 /* Valore sentinella per indicare la terminazione */

/* Struttura per rappresentare una matrice 3x3 */
typedef struct {
    int data[3][3];
    int is_poison;   /* Flag per la terminazione */
    int reader_id;   /* Per il log (opzionale, utile per debug) */
    int file_idx;    /* Indice del quadrato nel file originale */
} Matrix;

/* Struttura per il Buffer Circolare (Coda) */
typedef struct {
    Matrix *buffer;
    int size;
    int head;
    int tail;
    sem_t sem_empty;
    sem_t sem_full;
    pthread_mutex_t mutex;
} SafeQueue;

/* Struttura dati condivisa passata a tutti i thread */
typedef struct {
    SafeQueue intermediate_queue;
    SafeQueue final_queue;

    int num_readers_active;
    pthread_mutex_t mutex_readers_count; // Protegge il contatore dei lettori

    int num_verifiers_active;
    pthread_mutex_t mutex_verifiers_count; // Protegge il contatore dei verificatori

    pthread_mutex_t mutex_print;

    int M_verifiers;
    char **filenames;
} SharedData;

/* Struttura argomenti per i thread */
typedef struct {
    SharedData *shared;
```

```c
    int thread_id;     /* ID logico (1, 2, ...) */
    char *filename;    /* Solo per i lettori */
} ThreadArgs;

/* --- Funzioni di gestione Code --- */

void init_queue(SafeQueue *q, int size) {
    q->buffer = (Matrix *)malloc(sizeof(Matrix) * size);
    q->size = size;
    q->head = 0;
    q->tail = 0;
    sem_init(&q->sem_empty, 0, size);
    sem_init(&q->sem_full, 0, 0);
    pthread_mutex_init(&q->mutex, NULL);
}

void destroy_queue(SafeQueue *q) {
    free(q->buffer);
    sem_destroy(&q->sem_empty);
    sem_destroy(&q->sem_full);
    pthread_mutex_destroy(&q->mutex);
}

void insert_queue(SafeQueue *q, Matrix m) {
// 1. Aspetto che ci sia spazio (semaforo blocca se coda piena)
    sem_wait(&q->sem_empty);

    // 2. Entro nella "Sezione Critica" (solo uno alla volta)
    pthread_mutex_lock(&q->mutex);

    // 3. Inserisco il dato
    q->buffer[q->head] = m;
    q->head = (q->head + 1) % q->size; // Avanzamento circolare

    // 4. Esco dalla Sezione Critica
    pthread_mutex_unlock(&q->mutex);

    // 5. Avviso i consumatori che c'è un nuovo dato
    sem_post(&q->sem_full);
}

Matrix remove_queue(SafeQueue *q) {
    Matrix m;

    sem_wait(&q->sem_full); // Aspetto un dato

    pthread_mutex_lock(&q->mutex);

    m = q->buffer[q->tail];
    q->tail = (q->tail + 1) % q->size;

    pthread_mutex_unlock(&q->mutex);

    sem_post(&q->sem_empty); // Segnalo slot libero
```

```c
    return m;
}

/* --- Logica Quadrato Magico --- */

int is_magic(Matrix m, int *magic_total) {
    if (m.is_poison) return 0;

    int sum_diag1 = 0, sum_diag2 = 0;
    int i, j;

    // Calcolo diagonali
    for (i = 0; i < 3; i++) {
        sum_diag1 += m.data[i][i];
        sum_diag2 += m.data[i][2-i];
    }

    if (sum_diag1 != sum_diag2) return 0;
    *magic_total = sum_diag1;

    // Calcolo righe e colonne
    for (i = 0; i < 3; i++) {
        int row_sum = 0;
        int col_sum = 0;
        for (j = 0; j < 3; j++) {
            row_sum += m.data[i][j];
            col_sum += m.data[j][i];
        }
        if (row_sum != *magic_total || col_sum != *magic_total) return 0;
    }

    return 1;
}

void print_matrix_inline(Matrix m) {
    printf("(%d, %d, %d) (%d, %d, %d) (%d, %d, %d)\n",
        m.data[0][0], m.data[0][1], m.data[0][2],
        m.data[1][0], m.data[1][1], m.data[1][2],
        m.data[2][0], m.data[2][1], m.data[2][2]);
}

void print_magic_full(Matrix m, int total) {
    printf("[MAIN] quadrato magico trovato:\n");
    printf("(%d, %d, %d)\n", m.data[0][0], m.data[0][1], m.data[0][2]);
    printf("(%d, %d, %d)\n", m.data[1][0], m.data[1][1], m.data[1][2]);
    printf("(%d, %d, %d)\n", m.data[2][0], m.data[2][1], m.data[2][2]);
    printf("totale %d\n", total);
}

/* --- Thread Functions --- */

void *reader_routine(void *arg) {
    ThreadArgs *args = (ThreadArgs *)arg;
    SharedData *shared = args->shared;
```

```c
    FILE *f;
    char line[256];
    int count = 0;

    printf("[READER-%d] file '%s'\n", args->thread_id, args->filename);

    f = fopen(args->filename, "r");
    if (f) {
        while (fgets(line, sizeof(line), f)) {
            Matrix m;
            m.is_poison = 0;
            m.reader_id = args->thread_id;
            m.file_idx = ++count;

            // Parsing formato "36,13,27,9,24,41,18,3,30"
            // Nota: uso sscanf per semplicità, ma strtok_r sarebbe più robusto
            int parsed = sscanf(line, "%d,%d,%d,%d,%d,%d,%d,%d,%d",
                   &m.data[0][0], &m.data[0][1], &m.data[0][2],
                   &m.data[1][0], &m.data[1][1], &m.data[1][2],
                   &m.data[2][0], &m.data[2][1], &m.data[2][2]);

            if (parsed == 9) {
                // INIZIO SEZIONE CRITICA DI STAMPA
                pthread_mutex_lock(&shared->mutex_print);

                printf("[READER-%d] quadrato candidato n.%d: ", args->thread_id,
m.file_idx);
                print_matrix_inline(m); // La tua funzione di stampa

                pthread_mutex_unlock(&shared->mutex_print);
                // FINE SEZIONE CRITICA

                insert_queue(&shared->intermediate_queue, m);
            }
        }
        fclose(f);
    } else {
        perror("Errore apertura file");
    }

    // Gestione terminazione Lettori
    pthread_mutex_lock(&shared->mutex_readers_count);
    shared->num_readers_active--;
    if (shared->num_readers_active == 0) {
        // Sono l'ultimo lettore: mando pillole avvelenate ai verificatori
        int i;
        for (i = 0; i < shared->M_verifiers; i++) {
            Matrix poison;
            poison.is_poison = 1;
            insert_queue(&shared->intermediate_queue, poison);
        }
    }
    pthread_mutex_unlock(&shared->mutex_readers_count);

```

```c
    printf("[READER-%d] terminazione\n", args->thread_id);
    free(args);
    pthread_exit(NULL);
}

void *verifier_routine(void *arg) {
    ThreadArgs *args = (ThreadArgs *)arg;
    SharedData *shared = args->shared;
    int magic_tot;

    while (1) {
        Matrix m = remove_queue(&shared->intermediate_queue);

        if (m.is_poison) {
            // Ricevuta segnalazione di fine
            break;
        }

        pthread_mutex_lock(&shared->mutex_print);
        printf("[VERIF-%d] verifico quadrato: ", args->thread_id);
        print_matrix_inline(m);
        pthread_mutex_unlock(&shared->mutex_print);

        if (is_magic(m, &magic_tot)) {
            pthread_mutex_lock(&shared->mutex_print); // Proteggiamo anche l'annuncio
            printf("[VERIF-%d] trovato quadrato magico!\n", args->thread_id);
            pthread_mutex_unlock(&shared->mutex_print);

            insert_queue(&shared->final_queue, m);
        }
    }

    // Gestione terminazione Verificatori
    pthread_mutex_lock(&shared->mutex_verifiers_count);
    shared->num_verifiers_active--;
    if (shared->num_verifiers_active == 0) {
        // Sono l'ultimo verificatore: mando pillola al main
        Matrix poison;
        poison.is_poison = 1;
        insert_queue(&shared->final_queue, poison);
    }
    pthread_mutex_unlock(&shared->mutex_verifiers_count);

    printf("[VERIF-%d] terminazione\n", args->thread_id);
    free(args);
    pthread_exit(NULL);
}

/* --- MAIN --- */

int main(int argc, char *argv[]) {
    if (argc < 3) {
        fprintf(stderr, "Uso: %s <M-verifiers> <file-1> ... <file-N>\n", argv[0]);
        exit(1);
```

```c
    }

    int M = atoi(argv[1]);
    int N = argc - 2;
    int i;

    if (M <= 0) {
        fprintf(stderr, "Errore: M deve essere > 0\n");
        exit(1);
    }

    printf("[MAIN] creazione di %d thread lettori e %d thread verificatori\n", N, M);

    // Inizializzazione struttura condivisa
    SharedData shared;
    shared.M_verifiers = M;
    shared.num_readers_active = N;
    shared.num_verifiers_active = M;
    shared.filenames = &argv[2];

    init_queue(&shared.intermediate_queue, QUEUE_INTERMEDIATE_SIZE);
    init_queue(&shared.final_queue, QUEUE_FINAL_SIZE);

    pthread_mutex_init(&shared.mutex_readers_count, NULL);
    pthread_mutex_init(&shared.mutex_verifiers_count, NULL);
    pthread_mutex_init(&shared.mutex_print, NULL);

    pthread_t *readers = malloc(sizeof(pthread_t) * N);
    pthread_t *verifiers = malloc(sizeof(pthread_t) * M);

    // Avvio Lettori
    for (i = 0; i < N; i++) {
        ThreadArgs *args = malloc(sizeof(ThreadArgs));
        args->shared = &shared;
        args->thread_id = i + 1;
        args->filename = argv[2 + i];
        pthread_create(&readers[i], NULL, reader_routine, args);
    }

    // Avvio Verificatori
    for (i = 0; i < M; i++) {
        ThreadArgs *args = malloc(sizeof(ThreadArgs));
        args->shared = &shared;
        args->thread_id = i + 1;
        args->filename = NULL;
        pthread_create(&verifiers[i], NULL, verifier_routine, args);
    }

    // Loop del Main (Consumatore Finale)
    while (1) {
        Matrix m = remove_queue(&shared.final_queue);

        if (m.is_poison) {
            break;
```

```c
        }

        int total;
        is_magic(m, &total); // Ricalcolo solo per stampare il totale corretto
        pthread_mutex_lock(&shared.mutex_print);
        print_magic_full(m, total);
        pthread_mutex_unlock(&shared.mutex_print);
    }

    printf("[MAIN] terminazione\n");

    // Attesa terminazione thread
    for (i = 0; i < N; i++) pthread_join(readers[i], NULL);
    for (i = 0; i < M; i++) pthread_join(verifiers[i], NULL);

    // Cleanup
    free(readers);
    free(verifiers);
    destroy_queue(&shared.intermediate_queue);
    destroy_queue(&shared.final_queue);
    pthread_mutex_destroy(&shared.mutex_readers_count);
    pthread_mutex_destroy(&shared.mutex_verifiers_count);
    pthread_mutex_destroy(&shared.mutex_print);

    return 0;
}
```