

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

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <pthread.h>
4  #include <semaphore.h>
5  #include <string.h>
6  #include <unistd.h>
7
8  /* Costanti definite dalla traccia */
9  #define QUEUE_INTERMEDIATE_SIZE 10
10 #define QUEUE_FINAL_SIZE 3
11 #define POISON_PILL -1 /* Valore sentinella per indicare la terminazione */
12
13 /* Struttura per rappresentare una matrice 3x3 */
14 typedef struct {
15     int data[3][3];
16     int is_poison; /* Flag per la terminazione */
17     int reader_id; /* Per il log (opzionale, utile per debug) */
18     int file_idx; /* Indice del quadrato nel file originale */
19 } Matrix;
20
21 /* Struttura per il Buffer Circolare (Coda) */
22 typedef struct {
23     Matrix *buffer;
24     int size;
25     int head;
26     int tail;
27     sem_t sem_empty;
28     sem_t sem_full;
29     pthread_mutex_t mutex;
30 } SafeQueue;
31
32 /* Struttura dati condivisa passata a tutti i thread */
33 typedef struct {
34     SafeQueue intermediate_queue;
35     SafeQueue final_queue;
36
37     int num_readers_active;
38     pthread_mutex_t mutex_readers_count; // Protegge il contatore dei lettori
39
40     int num_verifiers_active;
41     pthread_mutex_t mutex_verifiers_count; // Protegge il contatore dei verificatori
42
43     pthread_mutex_t mutex_print;
44
45     int M_verifiers;
46     char **filenames;
47 } SharedData;
48
49 /* Struttura argomenti per i thread */
50 typedef struct {
51     SharedData *shared;
```

```
52     int thread_id;      /* ID logico (1, 2, ...) */
53     char *filename;     /* Solo per i lettori */
54 } ThreadArgs;
55
56 /* --- Funzioni di gestione Code --- */
57
58 void init_queue(SafeQueue *q, int size) {
59     q->buffer = (Matrix *)malloc(sizeof(Matrix) * size);
60     q->size = size;
61     q->head = 0;
62     q->tail = 0;
63     sem_init(&q->sem_empty, 0, size);
64     sem_init(&q->sem_full, 0, 0);
65     pthread_mutex_init(&q->mutex, NULL);
66 }
67
68 void destroy_queue(SafeQueue *q) {
69     free(q->buffer);
70     sem_destroy(&q->sem_empty);
71     sem_destroy(&q->sem_full);
72     pthread_mutex_destroy(&q->mutex);
73 }
74
75 void insert_queue(SafeQueue *q, Matrix m) {
76 // 1. Aspetto che ci sia spazio (semaforo blocca se coda piena)
77     sem_wait(&q->sem_empty);
78
79     // 2. Entro nella "Sezione Critica" (solo uno alla volta)
80     pthread_mutex_lock(&q->mutex);
81
82     // 3. Inserisco il dato
83     q->buffer[q->head] = m;
84     q->head = (q->head + 1) % q->size; // Avanzamento circolare
85
86     // 4. Esco dalla Sezione Critica
87     pthread_mutex_unlock(&q->mutex);
88
89     // 5. Avviso i consumatori che c'è un nuovo dato
90     sem_post(&q->sem_full);
91 }
92
93 Matrix remove_queue(SafeQueue *q) {
94     Matrix m;
95
96     sem_wait(&q->sem_full); // Aspetto un dato
97
98     pthread_mutex_lock(&q->mutex);
99
100     m = q->buffer[q->tail];
101     q->tail = (q->tail + 1) % q->size;
102
103     pthread_mutex_unlock(&q->mutex);
104
105     sem_post(&q->sem_empty); // Segnalo slot libero
```

```
106     return m;
107 }
108
109 /* --- Logica Quadrato Magico --- */
110
111 int is_magic(Matrix m, int *magic_total) {
112     if (m.is_poison) return 0;
113
114     int sum_diag1 = 0, sum_diag2 = 0;
115     int i, j;
116
117     // Calcolo diagonali
118     for (i = 0; i < 3; i++) {
119         sum_diag1 += m.data[i][i];
120         sum_diag2 += m.data[i][2-i];
121     }
122
123     if (sum_diag1 != sum_diag2) return 0;
124     *magic_total = sum_diag1;
125
126     // Calcolo righe e colonne
127     for (i = 0; i < 3; i++) {
128         int row_sum = 0;
129         int col_sum = 0;
130         for (j = 0; j < 3; j++) {
131             row_sum += m.data[i][j];
132             col_sum += m.data[j][i];
133         }
134         if (row_sum != *magic_total || col_sum != *magic_total) return 0;
135     }
136
137     return 1;
138 }
139
140 void print_matrix_inline(Matrix m) {
141     printf("(%d, %d, %d) (%d, %d, %d) (%d, %d, %d)\n",
142         m.data[0][0], m.data[0][1], m.data[0][2],
143         m.data[1][0], m.data[1][1], m.data[1][2],
144         m.data[2][0], m.data[2][1], m.data[2][2]);
145 }
146
147 void print_magic_full(Matrix m, int total) {
148     printf("[MAIN] quadrato magico trovato:\n");
149     printf("(%d, %d, %d)\n", m.data[0][0], m.data[0][1], m.data[0][2]);
150     printf("(%d, %d, %d)\n", m.data[1][0], m.data[1][1], m.data[1][2]);
151     printf("(%d, %d, %d)\n", m.data[2][0], m.data[2][1], m.data[2][2]);
152     printf("totale %d\n", total);
153 }
154
155 /* --- Thread Functions --- */
156
157 void *reader_routine(void *arg) {
158     ThreadArgs *args = (ThreadArgs *)arg;
159     SharedData *shared = args->shared;
```

```
160 FILE *f;
161 char line[256];
162 int count = 0;
163
164 printf("[READER-%d] file '%s'\n", args->thread_id, args->filename);
165
166 f = fopen(args->filename, "r");
167 if (f) {
168     while (fgets(line, sizeof(line), f)) {
169         Matrix m;
170         m.is_poison = 0;
171         m.reader_id = args->thread_id;
172         m.file_idx = ++count;
173
174         // Parsing formato "36,13,27,9,24,41,18,3,30"
175         // Nota: uso sscanf per semplicità, ma strtok_r sarebbe più robusto
176         int parsed = sscanf(line, "%d,%d,%d,%d,%d,%d,%d,%d,%d",
177             &m.data[0][0], &m.data[0][1], &m.data[0][2],
178             &m.data[1][0], &m.data[1][1], &m.data[1][2],
179             &m.data[2][0], &m.data[2][1], &m.data[2][2]);
180
181         if (parsed == 9) {
182             // INIZIO SEZIONE CRITICA DI STAMPA
183             pthread_mutex_lock(&shared->mutex_print);
184
185             printf("[READER-%d] quadrato candidato n.%d: ", args->thread_id,
186 m.file_idx);
187
188             print_matrix_inline(m); // La tua funzione di stampa
189
190             pthread_mutex_unlock(&shared->mutex_print);
191             // FINE SEZIONE CRITICA
192
193             insert_queue(&shared->intermediate_queue, m);
194         }
195     }
196     fclose(f);
197 } else {
198     perror("Errore apertura file");
199 }
200
201 // Gestione terminazione Lettori
202 pthread_mutex_lock(&shared->mutex_readers_count);
203 shared->num_readers_active--;
204 if (shared->num_readers_active == 0) {
205     // Sono l'ultimo lettore: mando pillole avvelenate ai verificatori
206     int i;
207     for (i = 0; i < shared->M_verifiers; i++) {
208         Matrix poison;
209         poison.is_poison = 1;
210         insert_queue(&shared->intermediate_queue, poison);
211     }
212 }
213 pthread_mutex_unlock(&shared->mutex_readers_count);
```

```
213     printf("[READER-%d] terminazione\n", args->thread_id);
214     free(args);
215     pthread_exit(NULL);
216 }
217
218 void *verifier_routine(void *arg) {
219     ThreadArgs *args = (ThreadArgs *)arg;
220     SharedData *shared = args->shared;
221     int magic_tot;
222
223     while (1) {
224         Matrix m = remove_queue(&shared->intermediate_queue);
225
226         if (m.is_poison) {
227             // Ricevuta segnalazione di fine
228             break;
229         }
230
231         pthread_mutex_lock(&shared->mutex_print);
232         printf("[VERIF-%d] verifico quadrato: ", args->thread_id);
233         print_matrix_inline(m);
234         pthread_mutex_unlock(&shared->mutex_print);
235
236         if (is_magic(m, &magic_tot)) {
237             pthread_mutex_lock(&shared->mutex_print); // Proteggiamo anche l'annuncio
238             printf("[VERIF-%d] trovato quadrato magico!\n", args->thread_id);
239             pthread_mutex_unlock(&shared->mutex_print);
240
241             insert_queue(&shared->final_queue, m);
242         }
243     }
244
245     // Gestione terminazione Verificatori
246     pthread_mutex_lock(&shared->mutex_verifiers_count);
247     shared->num_verifiers_active--;
248     if (shared->num_verifiers_active == 0) {
249         // Sono l'ultimo verificatore: mando pillola al main
250         Matrix poison;
251         poison.is_poison = 1;
252         insert_queue(&shared->final_queue, poison);
253     }
254     pthread_mutex_unlock(&shared->mutex_verifiers_count);
255
256     printf("[VERIF-%d] terminazione\n", args->thread_id);
257     free(args);
258     pthread_exit(NULL);
259 }
260
261 /* --- MAIN --- */
262
263 int main(int argc, char *argv[]) {
264     if (argc < 3) {
265         fprintf(stderr, "Uso: %s <M-verifiers> <file-1> ... <file-N>\n", argv[0]);
266         exit(1);
267     }
268 }
```

```
267     }
268
269     int M = atoi(argv[1]);
270     int N = argc - 2;
271     int i;
272
273     if (M <= 0) {
274         fprintf(stderr, "Errore: M deve essere > 0\n");
275         exit(1);
276     }
277
278     printf("[MAIN] creazione di %d thread lettori e %d thread verificatori\n", N, M);
279
280     // Inizializzazione struttura condivisa
281     SharedData shared;
282     shared.M_verifiers = M;
283     shared.num_readers_active = N;
284     shared.num_verifiers_active = M;
285     shared.filenamees = &argv[2];
286
287     init_queue(&shared.intermediate_queue, QUEUE_INTERMEDIATE_SIZE);
288     init_queue(&shared.final_queue, QUEUE_FINAL_SIZE);
289
290     pthread_mutex_init(&shared.mutex_readers_count, NULL);
291     pthread_mutex_init(&shared.mutex_verifiers_count, NULL);
292     pthread_mutex_init(&shared.mutex_print, NULL);
293
294     pthread_t *readers = malloc(sizeof(pthread_t) * N);
295     pthread_t *verifiers = malloc(sizeof(pthread_t) * M);
296
297     // Avvio Lettori
298     for (i = 0; i < N; i++) {
299         ThreadArgs *args = malloc(sizeof(ThreadArgs));
300         args->shared = &shared;
301         args->thread_id = i + 1;
302         args->filename = argv[2 + i];
303         pthread_create(&readers[i], NULL, reader_routine, args);
304     }
305
306     // Avvio Verificatori
307     for (i = 0; i < M; i++) {
308         ThreadArgs *args = malloc(sizeof(ThreadArgs));
309         args->shared = &shared;
310         args->thread_id = i + 1;
311         args->filename = NULL;
312         pthread_create(&verifiers[i], NULL, verifier_routine, args);
313     }
314
315     // Loop del Main (Consumatore Finale)
316     while (1) {
317         Matrix m = remove_queue(&shared.final_queue);
318
319         if (m.is_poison) {
320             break;
```

```
321     }
322
323     int total;
324     is_magic(m, &total); // Ricalcolo solo per stampare il totale corretto
325     pthread_mutex_lock(&shared.mutex_print);
326     print_magic_full(m, total);
327     pthread_mutex_unlock(&shared.mutex_print);
328 }
329
330 printf("[MAIN] terminazione\n");
331
332 // Attesa terminazione thread
333 for (i = 0; i < N; i++) pthread_join(readers[i], NULL);
334 for (i = 0; i < M; i++) pthread_join(verifiers[i], NULL);
335
336 // Cleanup
337 free(readers);
338 free(verifiers);
339 destroy_queue(&shared.intermediate_queue);
340 destroy_queue(&shared.final_queue);
341 pthread_mutex_destroy(&shared.mutex_readers_count);
342 pthread_mutex_destroy(&shared.mutex_verifiers_count);
343 pthread_mutex_destroy(&shared.mutex_print);
344
345 return 0;
346 }
```