D:\Lab_Sistemi-Operativi\Tutorato-Sistemi-Operativi-2025-main\Esame-[2025-07-17]\gemini_semi_magic_squares.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <pthread.h>
#include <string.h>

#define MAX_M 16
#define CAPIENZA_CODA 5

/* * STRUTTURE DATI
 */

/* Rappresenta una singola matrice letta dal file */
typedef struct {
    unsigned char dati[MAX_M][MAX_M];
    int id_lettore;   /* Per output: [READER-ID] */
    int n_sequenza;   /* Per output: quadrato n.X */
    int size_m;       /* Dimensione M */
} Matrice;

/* MONITOR 1: Coda Intermedia (Reader -> Verifier) */
typedef struct {
    Matrice buffer[CAPIENZA_CODA];
    int testa;
    int coda;
    int count;              /* Numero elementi attuali */
    int lettori_attivi;     /* Contatore per capire quando finiscono TUTTI i lettori */
    pthread_mutex_t mutex;
    pthread_cond_t non_pieno;
    pthread_cond_t non_vuoto;
} CodaIntermedia;

/* MONITOR 2: Buffer Singolo (Verifier -> Main) */
typedef struct {
    Matrice matrice;
    int piena;              /* 1 se c'è un dato da leggere, 0 altrimenti */
    int somma_magica;       /* Il risultato del calcolo */
    int finito;             /* Flag: 1 se il Verifier ha finito tutto */
    pthread_mutex_t mutex;
    pthread_cond_t main_ready; /* Main aspetta dato */
    pthread_cond_t verif_done; /* Verifier aspetta che Main legga */
} RecordFinale;

/* Struttura per passare parametri ai thread lettori */
typedef struct {
    int id;
    int M;
    char *nome_file;
```

```c
    CodaIntermedia *coda;
} ReaderArgs;

/* Struttura per passare parametri al thread verifier */
typedef struct {
    CodaIntermedia *coda_in;
    RecordFinale *record_out;
} VerifierArgs;


/*
 * FUNZIONI DI UTILITÀ
 */

/* Stampa una matrice nel formato richiesto: (a, b, c) (d, e, f) ... */
void stampa_matrice_lineare(Matrice *m) {
    for (int r = 0; r < m->size_m; r++) {
        printf("(");
        for (int c = 0; c < m->size_m; c++) {
            printf("%d", m->dati[r][c]);
            if (c < m->size_m - 1) printf(", ");
        }
        printf(")");
        if (r < m->size_m - 1) printf(" ");
    }
}

/* Verifica se semi-magica. Ritorna la somma se sì, -1 se no */
int verifica_semi_magico(Matrice *m) {
    int M = m->size_m;
    int somma_target = 0;

    /* Calcola somma prima riga come riferimento */
    for (int c = 0; c < M; c++) somma_target += m->dati[0][c];

    /* Verifica righe */
    for (int r = 1; r < M; r++) {
        int somma = 0;
        for (int c = 0; c < M; c++) somma += m->dati[r][c];
        if (somma != somma_target) return -1;
    }

    /* Verifica colonne */
    for (int c = 0; c < M; c++) {
        int somma = 0;
        for (int r = 0; r < M; r++) somma += m->dati[r][c];
        if (somma != somma_target) return -1;
    }

    return somma_target;
}

/*
 * THREAD LETTORE
```

```c
 */
void* reader_thread(void* arg) {
    ReaderArgs *args = (ReaderArgs*)arg;
    int M = args->M;

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

    int fd = open(args->nome_file, O_RDONLY);
    if (fd == -1) {
        perror("open");
        /* Decrementiamo comunque i lettori attivi per non bloccare il sistema */
        pthread_mutex_lock(&args->coda->mutex);
        args->coda->lettori_attivi--;
        pthread_cond_broadcast(&args->coda->non_vuoto);
        pthread_mutex_unlock(&args->coda->mutex);
        pthread_exit(NULL);
    }

    struct stat sb;
    if (fstat(fd, &sb) == -1) { perror("fstat"); close(fd); exit(1); }

    /* MAPPATURA MEMORIA */
    unsigned char* file_map = mmap(NULL, sb.st_size, PROT_READ, MAP_PRIVATE, fd, 0);
    if (file_map == MAP_FAILED) { perror("mmap"); close(fd); exit(1); }

    int num_matrici = sb.st_size / (M * M);

    for (int i = 0; i < num_matrici; i++) {
        Matrice m;
        m.id_lettore = args->id;
        m.n_sequenza = i + 1;
        m.size_m = M;

        /* Copia i byte dalla mappa di memoria alla struttura */
        /* Nota: i file binari sono lineari, copiamo riga per riga */
        unsigned char *base_addr = file_map + (i * M * M);
        for (int r=0; r<M; r++) {
            for(int c=0; c<M; c++) {
                m.dati[r][c] = base_addr[(r*M) + c];
            }
        }

        flockfile(stdout);

        /* Output richiesto PRIMA dell'inserimento */
        printf("[READER-%d] quadrato candidato n.%d: ", m.id_lettore, m.n_sequenza);
        stampa_matrice_lineare(&m);
        printf("\n");

        funlockfile(stdout);

        /* INSERIMENTO IN CODA (Produttore) */
        pthread_mutex_lock(&args->coda->mutex);
        while (args->coda->count == CAPIENZA_CODA) {
```

```
160             pthread_cond_wait(&args->coda->non_pieno, &args->coda->mutex);
161         }
162 
163         args->coda->buffer[args->coda->testa] = m;
164         args->coda->testa = (args->coda->testa + 1) % CAPIENZA_CODA;
165         args->coda->count++;
166 
167         pthread_cond_signal(&args->coda->non_vuoto);
168         pthread_mutex_unlock(&args->coda->mutex);
169     }
170 
171     /* Pulizia e Chiusura */
172     munmap(file_map, sb.st_size);
173     close(fd);
174 
175     printf("[READER-%d] terminazione con %d quadrati letti\n", args->id, num_matrici);
176 
177     /* Segnalazione Fine Lettore */
178     pthread_mutex_lock(&args->coda->mutex);
179     args->coda->lettori_attivi--;
180     /* Svegliamo il Verifier nel caso fosse bloccato su coda vuota */
181     pthread_cond_broadcast(&args->coda->non_vuoto);
182     pthread_mutex_unlock(&args->coda->mutex);
183 
184     return NULL;
185 }
186 
187 /*
188  * THREAD VERIFICATORE
189  */
190 void* verifier_thread(void* arg) {
191     VerifierArgs *args = (VerifierArgs*)arg;
192     CodaIntermedia *cin = args->coda_in;
193     RecordFinale *rout = args->record_out;
194 
195     while (1) {
196         Matrice m;
197 
198         /* PRELIEVO DALLA CODA (Consumatore) */
199         pthread_mutex_lock(&cin->mutex);
200 
201         /* Attendo se vuoto, MA controllo anche se i lettori sono finiti */
202         while (cin->count == 0 && cin->lettori_attivi > 0) {
203             pthread_cond_wait(&cin->non_vuoto, &cin->mutex);
204         }
205 
206         /* Se vuoto e nessun lettore attivo, ho finito */
207         if (cin->count == 0 && cin->lettori_attivi == 0) {
208             pthread_mutex_unlock(&cin->mutex);
209             break; /* ESCE DAL WHILE INFINITO */
210         }
211 
212         /* Prelievo */
213         m = cin->buffer[cin->coda];
```

```c
        cin->coda = (cin->coda + 1) % CAPIENZA_CODA;
        cin->count--;

        pthread_cond_signal(&cin->non_pieno);
        pthread_mutex_unlock(&cin->mutex);

        /* VERIFICA */
        printf("[VERIF] verifico quadrato: ");
        stampa_matrice_lineare(&m);
        printf("\n");

        int somma = verifica_semi_magico(&m);
        if (somma != -1) {
            printf("[VERIF] trovato quadrato semi-magico!\n");

            /* PASSA AL MAIN (Produttore su buffer singolo) */
            pthread_mutex_lock(&rout->mutex);
            while (rout->piena == 1) {
                pthread_cond_wait(&rout->verif_done, &rout->mutex);
            }

            rout->matrice = m;
            rout->somma_magica = somma;
            rout->piena = 1;

            pthread_cond_signal(&rout->main_ready);
            pthread_mutex_unlock(&rout->mutex);
        }
    }

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

    /* Segnala fine al Main */
    pthread_mutex_lock(&rout->mutex);
    rout->finito = 1;
    pthread_cond_signal(&rout->main_ready);
    pthread_mutex_unlock(&rout->mutex);

    return NULL;
}

/*
 * MAIN
 */
int main(int argc, char *argv[]) {
    if (argc < 3) {
        fprintf(stderr, "Uso: %s <M> <file1> [file2 ...]\n", argv[0]);
        exit(1);
    }

    int M = atoi(argv[1]);
    if (M < 3 || M > 16) {
        fprintf(stderr, "Errore: M deve essere tra 3 e 16\n");
        exit(1);
```

```c
    }

    int num_files = argc - 2;
    printf("[MAIN] creazione di %d thread lettori e 1 thread verificatore\n", num_files);

    /* Inizializzazione Strutture Condivise */
    CodaIntermedia coda;
    coda.testa = 0; coda.coda = 0; coda.count = 0;
    coda.lettori_attivi = num_files;
    pthread_mutex_init(&coda.mutex, NULL);
    pthread_cond_init(&coda.non_pieno, NULL);
    pthread_cond_init(&coda.non_vuoto, NULL);

    RecordFinale record_finale;
    record_finale.piena = 0;
    record_finale.finito = 0;
    pthread_mutex_init(&record_finale.mutex, NULL);
    pthread_cond_init(&record_finale.main_ready, NULL);
    pthread_cond_init(&record_finale.verif_done, NULL);

    /* Avvio Thread Verificatore */
    pthread_t verif_tid;
    VerifierArgs v_args;
    v_args.coda_in = &coda;
    v_args.record_out = &record_finale;
    pthread_create(&verif_tid, NULL, verifier_thread, &v_args);

    /* Avvio Thread Lettori */
    pthread_t *readers = malloc(num_files * sizeof(pthread_t));
    ReaderArgs *r_args = malloc(num_files * sizeof(ReaderArgs)); // Array di struct per
args

    for (int i = 0; i < num_files; i++) {
        r_args[i].id = i + 1;
        r_args[i].M = M;
        r_args[i].nome_file = argv[i + 2];
        r_args[i].coda = &coda;
        pthread_create(&readers[i], NULL, reader_thread, &r_args[i]);
    }

    /* CICLO MAIN (Consumatore finale) */
    int trovati = 0;
    while (1) {
        pthread_mutex_lock(&record_finale.mutex);

        while (record_finale.piena == 0 && record_finale.finito == 0) {
            pthread_cond_wait(&record_finale.main_ready, &record_finale.mutex);
        }

        if (record_finale.piena == 0 && record_finale.finito == 1) {
            pthread_mutex_unlock(&record_finale.mutex);
            break; /* Fine lavori */
        }

```

```c
        /* Stampa risultato */
        printf("[MAIN] quadrato semi-magico trovato:\n");
        stampa_matrice_lineare(&record_finale.matrice);
        printf("\n");
        printf("totale semi-magico %d\n", record_finale.somma_magica);

        trovati++;
        record_finale.piena = 0; // Segna come letto

        pthread_cond_signal(&record_finale.verif_done);
        pthread_mutex_unlock(&record_finale.mutex);
    }

    printf("[MAIN] terminazione con %d quadrati semi-magici trovati\n", trovati);

    /* Attesa terminazione thread (Join) */
    for (int i = 0; i < num_files; i++) {
        pthread_join(readers[i], NULL);
    }
    pthread_join(verif_tid, NULL);

    /* Pulizia */
    free(readers);
    free(r_args);
    pthread_mutex_destroy(&coda.mutex);
    pthread_cond_destroy(&coda.non_pieno);
    pthread_cond_destroy(&coda.non_vuoto);
    pthread_mutex_destroy(&record_finale.mutex);
    pthread_cond_destroy(&record_finale.main_ready);
    pthread_cond_destroy(&record_finale.verif_done);

    return 0;
}
```