D:\Lab_Sistemi-Operativi\Tutorato-Sistemi-Operativi-2025-main\Esame-[2025-09-17]\gemini_equisum_mmap.c

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

#define M 12
#define BUFFER_SIZE 10
#define NUM_VERIFIERS 3

#define LOCK_IO()   flockfile(stdout)
#define UNLOCK_IO() funlockfile(stdout)

// ==================== STRUTTURE ====================

typedef struct {
    unsigned char dati[M];
    int file_id;
    int vector_idx;
} Record;

typedef struct {
    // Coda Intermedia
    Record buffer[BUFFER_SIZE];
    int in, out, count;
    bool buffer_closed;
    pthread_mutex_t mtx_buffer;
    pthread_cond_t cond_buf_not_empty;
    pthread_cond_t cond_buf_not_full;

    // Slot Finale
    Record slot_finale;
    bool slot_full;
    bool slot_closed;
    pthread_mutex_t mtx_slot;
    pthread_cond_t cond_slot_empty;
    pthread_cond_t cond_slot_full;

    // Contatori per gestione chiusura automatica
    int readers_active;
    int verifiers_active;
    pthread_mutex_t mtx_counts;

} SharedData;

// ==================== HELPER ====================
void print_vec(unsigned char *v) {
```

```c
    for (int i = 0; i < M; i++) {
        printf("%u", v[i]);
        if (i < M - 1) printf(", ");
    }
}

// ==================== GESTIONE SHARED ====================
void init_shared(SharedData *s, int n_readers) {
    s->in = 0; s->out = 0; s->count = 0; s->buffer_closed = false;
    pthread_mutex_init(&s->mtx_buffer, NULL);
    pthread_cond_init(&s->cond_buf_not_empty, NULL);
    pthread_cond_init(&s->cond_buf_not_full, NULL);

    s->slot_full = false; s->slot_closed = false;
    pthread_mutex_init(&s->mtx_slot, NULL);
    pthread_cond_init(&s->cond_slot_empty, NULL);
    pthread_cond_init(&s->cond_slot_full, NULL);

    s->readers_active = n_readers;
    s->verifiers_active = NUM_VERIFIERS;
    pthread_mutex_init(&s->mtx_counts, NULL);
}

// Funzioni Buffer
void buffer_put(SharedData *s, Record r) {
    pthread_mutex_lock(&s->mtx_buffer);
    while (s->count == BUFFER_SIZE && !s->buffer_closed)
        pthread_cond_wait(&s->cond_buf_not_full, &s->mtx_buffer);
    if (!s->buffer_closed) {
        s->buffer[s->in] = r;
        s->in = (s->in + 1) % BUFFER_SIZE;
        s->count++;
        pthread_cond_signal(&s->cond_buf_not_empty);
    }
    pthread_mutex_unlock(&s->mtx_buffer);
}

bool buffer_get(SharedData *s, Record *r) {
    pthread_mutex_lock(&s->mtx_buffer);
    while (s->count == 0 && !s->buffer_closed)
        pthread_cond_wait(&s->cond_buf_not_empty, &s->mtx_buffer);
    bool res = false;
    if (s->count > 0) {
        *r = s->buffer[s->out];
        s->out = (s->out + 1) % BUFFER_SIZE;
        s->count--;
        pthread_cond_signal(&s->cond_buf_not_full);
        res = true;
    }
    pthread_mutex_unlock(&s->mtx_buffer);
    return res;
}

void buffer_close(SharedData *s) {
```

```
106     pthread_mutex_lock(&s->mtx_buffer);
107     s->buffer_closed = true;
108     pthread_cond_broadcast(&s->cond_buf_not_empty);
109     pthread_cond_broadcast(&s->cond_buf_not_full);
110     pthread_mutex_unlock(&s->mtx_buffer);
111 }
112 
113 // Funzioni Slot Finale
114 void slot_put(SharedData *s, Record r) {
115     pthread_mutex_lock(&s->mtx_slot);
116     while (s->slot_full && !s->slot_closed)
117         pthread_cond_wait(&s->cond_slot_empty, &s->mtx_slot);
118     if (!s->slot_closed) {
119         s->slot_finale = r;
120         s->slot_full = true;
121         pthread_cond_signal(&s->cond_slot_full);
122     }
123     pthread_mutex_unlock(&s->mtx_slot);
124 }
125 
126 bool slot_get(SharedData *s, Record *r) {
127     pthread_mutex_lock(&s->mtx_slot);
128     while (!s->slot_full && !s->slot_closed)
129         pthread_cond_wait(&s->cond_slot_full, &s->mtx_slot);
130     bool res = false;
131     if (s->slot_full) {
132         *r = s->slot_finale;
133         s->slot_full = false;
134         pthread_cond_signal(&s->cond_slot_empty);
135         res = true;
136     }
137     pthread_mutex_unlock(&s->mtx_slot);
138     return res;
139 }
140 
141 void slot_close(SharedData *s) {
142     pthread_mutex_lock(&s->mtx_slot);
143     s->slot_closed = true;
144     pthread_cond_broadcast(&s->cond_slot_full);
145     pthread_cond_broadcast(&s->cond_slot_empty);
146     pthread_mutex_unlock(&s->mtx_slot);
147 }
148 
149 // ===================== THREADS =====================
150 
151 typedef struct { SharedData *S; char *filename; int id; } ReaderArgs;
152 typedef struct { SharedData *S; int id; } VerifierArgs;
153 
154 void *reader_thread(void *arg) {
155     ReaderArgs *a = (ReaderArgs *)arg;
156 
157     LOCK_IO(); printf("[READER-%d] file '%s'\n", a->id, a->filename); UNLOCK_IO();
158 
159     int fd = open(a->filename, O_RDONLY);
```

```c
    if (fd == -1) { perror("open"); return NULL; } // Gestione errore base
    struct stat sb; fstat(fd, &sb);
    size_t fsize = sb.st_size;
    unsigned char *map = mmap(NULL, fsize, PROT_READ, MAP_PRIVATE, fd, 0);
    close(fd);

    int num_vec = fsize / M;
    for (int i = 0; i < num_vec; i++) {
        Record r;
        memcpy(r.dati, map + (i*M), M);
        r.file_id = a->id; r.vector_idx = i + 1;

        LOCK_IO();
        printf("[READER-%d] vettore candidato n.%d: ", a->id, i + 1);
        print_vec(r.dati);
        printf("\n");
        UNLOCK_IO();

        buffer_put(a->S, r);
    }
    if (map != MAP_FAILED) munmap(map, fsize);

    LOCK_IO(); printf("[READER-%d] terminazione con %d vettori letti\n", a->id, num_vec);
UNLOCK_IO();

    // Logica Last Man Standing per i lettori
    pthread_mutex_lock(&a->S->mtx_counts);
    a->S->readers_active--;
    if (a->S->readers_active == 0) buffer_close(a->S);
    pthread_mutex_unlock(&a->S->mtx_counts);

    return NULL;
}

void *verifier_thread(void *arg) {
    VerifierArgs *a = (VerifierArgs *)arg;
    Record r;
    int count = 0;

    while (buffer_get(a->S, &r)) {
        count++;
        int p = 0, d = 0;
        for (int i=0; i<M; i++) { if(i%2==0) p+=r.dati[i]; else d+=r.dati[i]; }

        LOCK_IO();
        printf("[VERIF-%d] verifico vettore: ", a->id);
        print_vec(r.dati);
        printf("\n");
        if (p == d) {
            printf("[VERIF-%d] si tratta di un vettore equisomma con somma %d!\n", a->id,
p);
            UNLOCK_IO(); // Sblocco prima di put per evitare attese con lock output
            slot_put(a->S, r);
        } else {
```

```c
            printf("[VERIF-%d] non è un vettore equisomma (somma pari %d vs. dispari %d)\n", a->id, p, d);
            UNLOCK_IO();
        }
    }

    LOCK_IO(); printf("[VERIF-%d] terminazione con %d vettori verificati\n", a->id, count); UNLOCK_IO();

    // Logica Last Man Standing per i verificatori
    pthread_mutex_lock(&a->S->mtx_counts);
    a->S->verifiers_active--;
    if (a->S->verifiers_active == 0) slot_close(a->S);
    pthread_mutex_unlock(&a->S->mtx_counts);

    free(a);
    return NULL;
}

int main(int argc, char *argv[]) {
    if (argc < 2) { printf("Uso: %s <f1> ...\n", argv[0]); return 1; }
    int n_readers = argc - 1;
    SharedData S;
    init_shared(&S, n_readers);

    LOCK_IO();
    printf("[MAIN] creazione di %d thread lettori e %d thread verificatori\n", n_readers, NUM_VERIFIERS);
    UNLOCK_IO();

    pthread_t readers[n_readers], verifiers[NUM_VERIFIERS];
    ReaderArgs r_args[n_readers];

    for (int i=0; i<n_readers; i++) {
        r_args[i].S = &S; r_args[i].filename = argv[i+1]; r_args[i].id = i+1;
        pthread_create(&readers[i], NULL, reader_thread, &r_args[i]);
    }
    for (int i=0; i<NUM_VERIFIERS; i++) {
        VerifierArgs *va = malloc(sizeof(VerifierArgs));
        va->S = &S; va->id = i+1;
        pthread_create(&verifiers[i], NULL, verifier_thread, va);
    }

    // Il Main consuma i risultati
    Record r;
    int total = 0;
    while (slot_get(&S, &r)) {
        total++;
        LOCK_IO();
        printf("[MAIN] ricevuto nuovo vettore equisomma: ");
        print_vec(r.dati);
        printf("\n");
        UNLOCK_IO();
    }
```

```
263
264     for(int i=0;i<n_readers;i++) pthread_join(readers[i], NULL);
265     for(int i=0;i<NUM_VERIFIERS;i++) pthread_join(verifiers[i], NULL);
266
267     LOCK_IO(); printf("[MAIN] terminazione con %d vettori equisomma trovati\n", total);
    UNLOCK_IO();
268     return 0;
269 }
```