ĐẠI HỌC QUỐC GIA TP.HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC BÁCH KHOA

KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

BÁO CÁO

LAB 3 : SYNCHRONIZATION

GIẢNG VIÊN HƯỚNG DẪN: BÙI XUÂN GIANG

SINH VIÊN THỰC HIỆN: NGUYỄN TẤN PHÁT

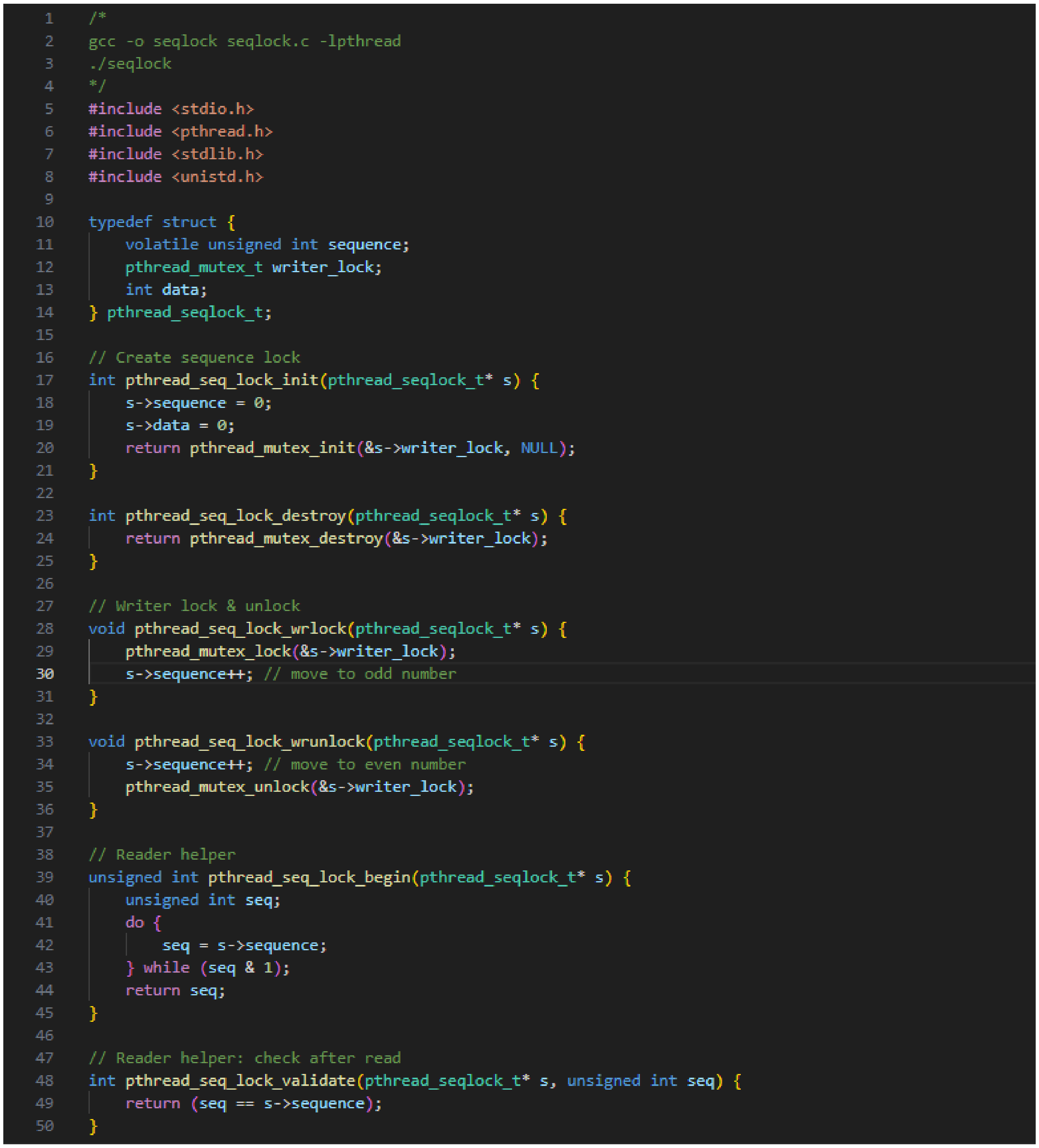
MSSV: 2352888

LỚP: CN01

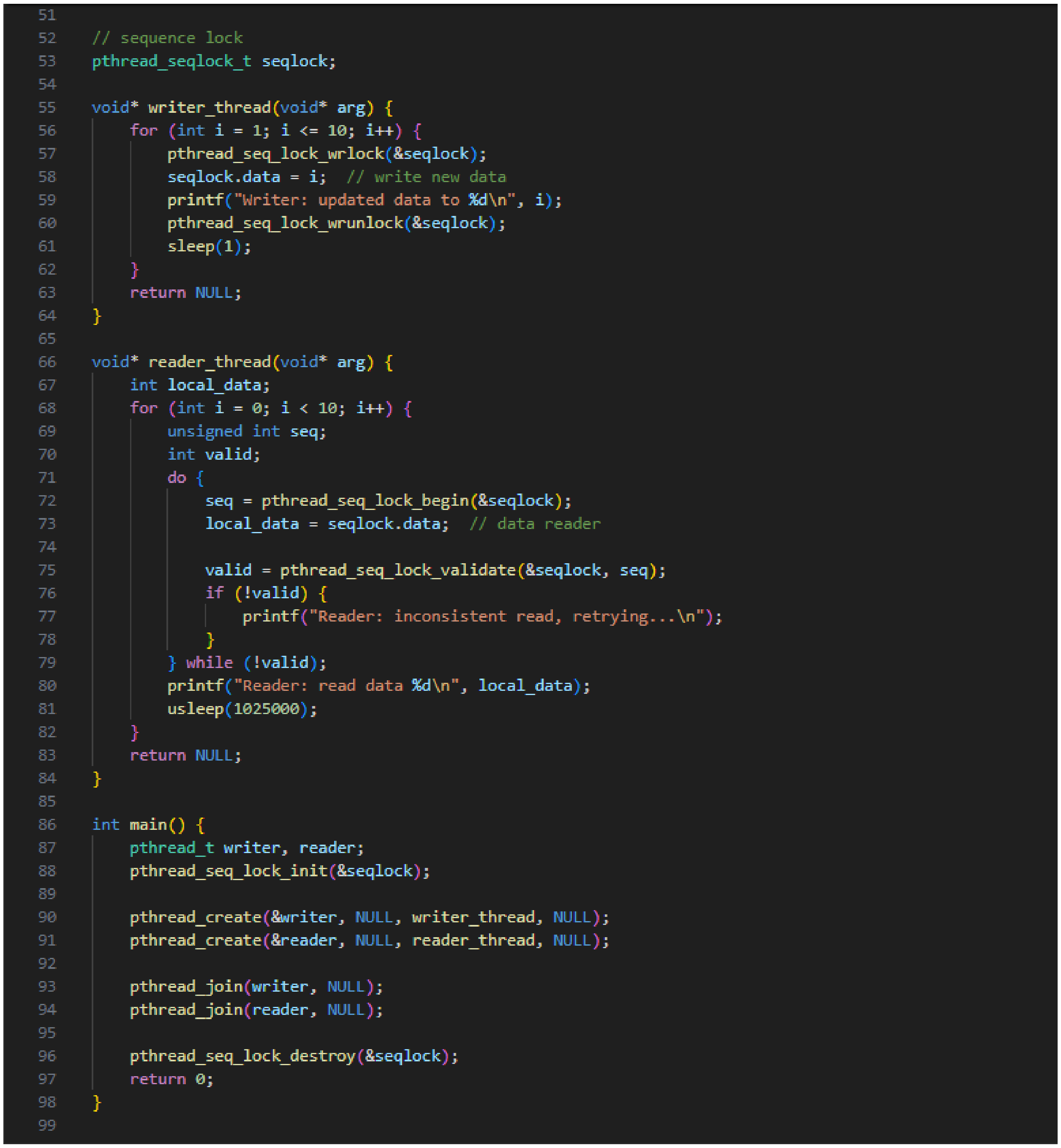
Thành phố Hồ Chí Minh – 2025

PROBLEM 1

Code:

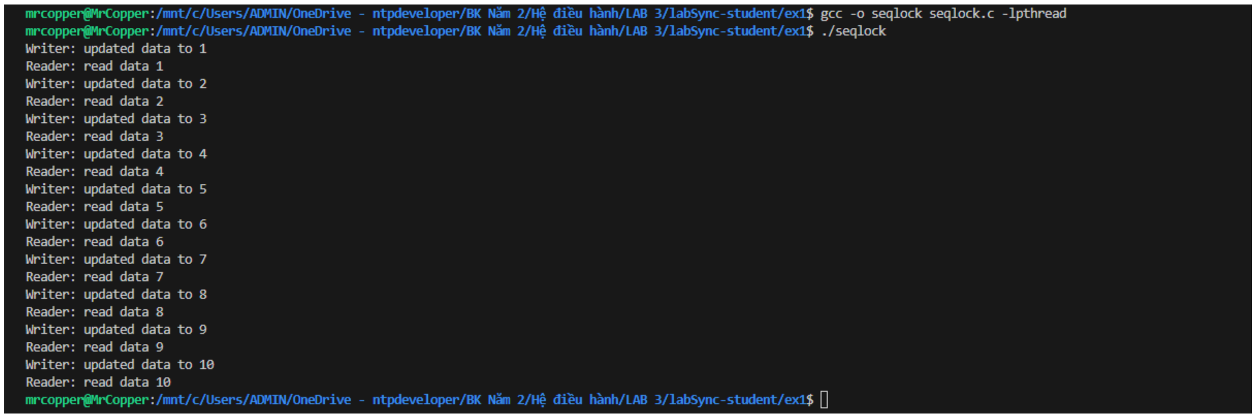


```c
/*
gcc -o seqlock seqlock.c -lpthread
./seqlock
*/
#include <stdio.h>
#include <pthread.h>
#include <stdlib.h>
#include <unistd.h>

typedef struct {
    volatile unsigned int sequence;
    pthread_mutex_t writer_lock;
    int data;
} pthread_seqlock_t;

// Create sequence lock
int pthread_seq_lock_init(pthread_seqlock_t* s) {
    s->sequence = 0;
    s->data = 0;
    return pthread_mutex_init(&s->writer_lock, NULL);
}

int pthread_seq_lock_destroy(pthread_seqlock_t* s) {
    return pthread_mutex_destroy(&s->writer_lock);
}

// Writer lock & unlock
void pthread_seq_lock_wrlock(pthread_seqlock_t* s) {
    pthread_mutex_lock(&s->writer_lock);
    s->sequence++; // move to odd number
}

void pthread_seq_lock_wrunlock(pthread_seqlock_t* s) {
    s->sequence++; // move to even number
    pthread_mutex_unlock(&s->writer_lock);
}

// Reader helper
unsigned int pthread_seq_lock_begin(pthread_seqlock_t* s) {
    unsigned int seq;
    do {
        seq = s->sequence;
    } while (seq & 1);
    return seq;
}

// Reader helper: check after read
int pthread_seq_lock_validate(pthread_seqlock_t* s, unsigned int seq) {
    return (seq == s->sequence);
}
```



```c

// sequence lock
pthread_seqlock_t seqlock;

void* writer_thread(void* arg) {
    for (int i = 1; i <= 10; i++) {
        pthread_seq_lock_wrlock(&seqlock);
        seqlock.data = i;  // write new data
        printf("Writer: updated data to %d\n", i);
        pthread_seq_lock_wrunlock(&seqlock);
        sleep(1);
    }
    return NULL;
}

void* reader_thread(void* arg) {
    int local_data;
    for (int i = 0; i < 10; i++) {
        unsigned int seq;
        int valid;
        do {
            seq = pthread_seq_lock_begin(&seqlock);
            local_data = seqlock.data;  // data reader

            valid = pthread_seq_lock_validate(&seqlock, seq);
            if (!valid) {
                printf("Reader: inconsistent read, retrying...\n");
            }
        } while (!valid);
        printf("Reader: read data %d\n", local_data);
        usleep(1025000);
    }
    return NULL;
}

int main() {
    pthread_t writer, reader;
    pthread_seq_lock_init(&seqlock);

    pthread_create(&writer, NULL, writer_thread, NULL);
    pthread_create(&reader, NULL, reader_thread, NULL);

    pthread_join(writer, NULL);
    pthread_join(reader, NULL);

    pthread_seq_lock_destroy(&seqlock);
    return 0;
}

```

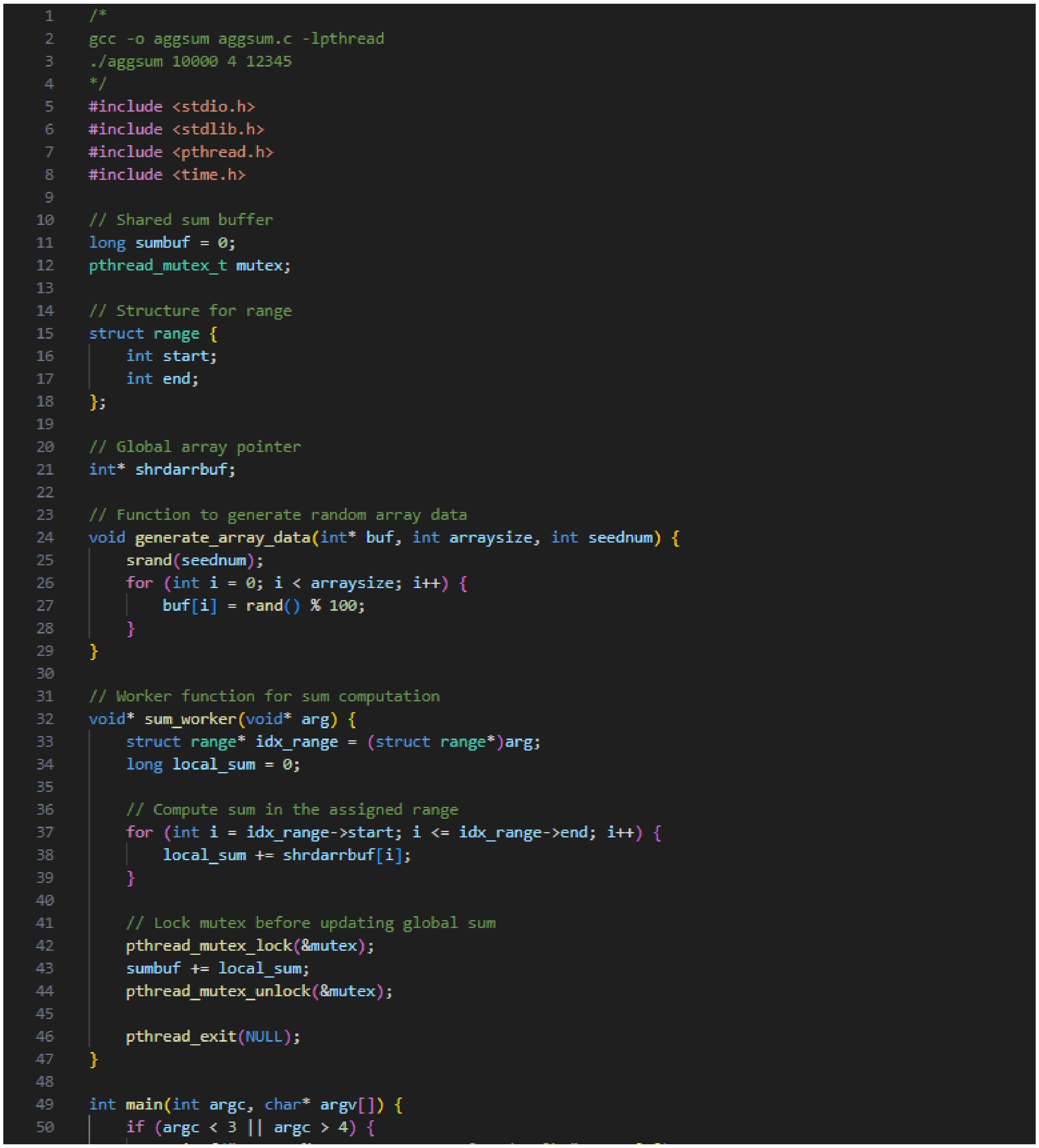
Output:



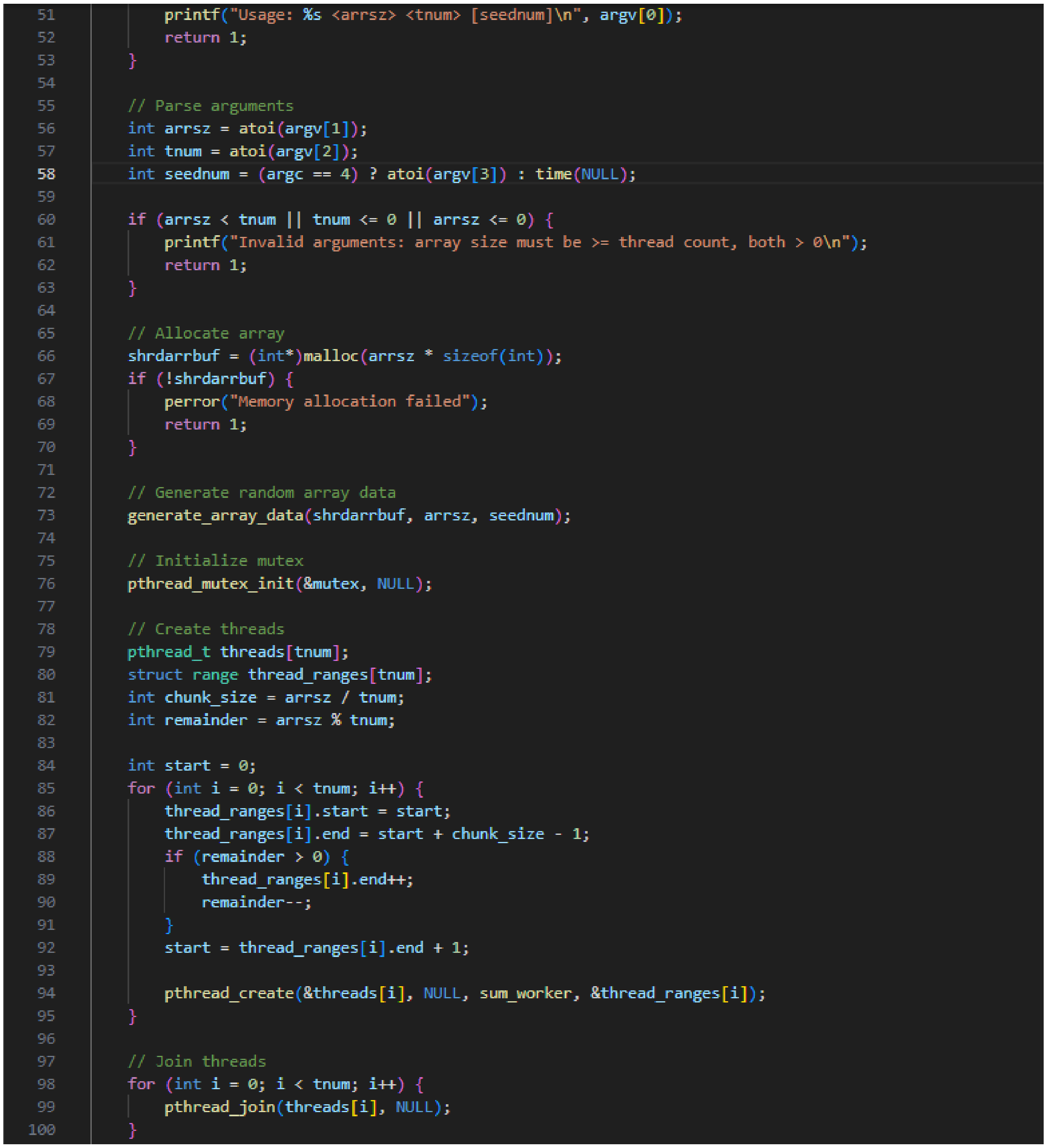
```
mrcopper@MrCopper:/mnt/c/Users/ADMIN/OneDrive - ntpdeveloper/BK Năm 2/Hệ điều hành/LAB 3/labSync-student/ex1$ gcc -o seqlock seqlock.c -lpthread
mrcopper@MrCopper:/mnt/c/Users/ADMIN/OneDrive - ntpdeveloper/BK Năm 2/Hệ điều hành/LAB 3/labSync-student/ex1$ ./seqlock
Writer: updated data to 1
Reader: read data 1
Writer: updated data to 2
Reader: read data 2
Writer: updated data to 3
Reader: read data 3
Writer: updated data to 4
Reader: read data 4
Writer: updated data to 5
Reader: read data 5
Writer: updated data to 6
Reader: read data 6
Writer: updated data to 7
Reader: read data 7
Writer: updated data to 8
Reader: read data 8
Writer: updated data to 9
Reader: read data 9
Writer: updated data to 10
Reader: read data 10
mrcopper@MrCopper:/mnt/c/Users/ADMIN/OneDrive - ntpdeveloper/BK Năm 2/Hệ điều hành/LAB 3/labSync-student/ex1$ 
```

PROBLEM 2

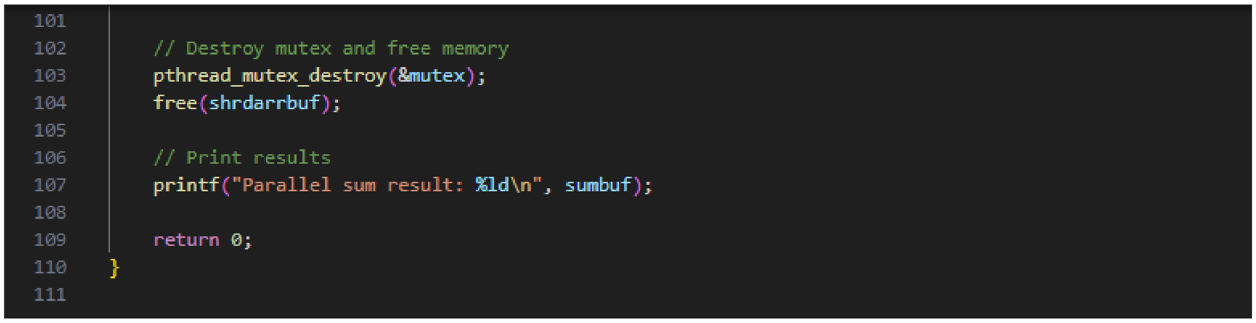
Code:



```c
/*
gcc -o aggsum aggsum.c -lpthread
./aggsum 10000 4 12345
*/
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <time.h>

// Shared sum buffer
long sumbuf = 0;
pthread_mutex_t mutex;

// Structure for range
struct range {
    int start;
    int end;
};

// Global array pointer
int* shrdarrbuf;

// Function to generate random array data
void generate_array_data(int* buf, int arraysize, int seednum) {
    srand(seednum);
    for (int i = 0; i < arraysize; i++) {
        buf[i] = rand() % 100;
    }
}

// Worker function for sum computation
void* sum_worker(void* arg) {
    struct range* idx_range = (struct range*)arg;
    long local_sum = 0;

    // Compute sum in the assigned range
    for (int i = idx_range->start; i <= idx_range->end; i++) {
        local_sum += shrdarrbuf[i];
    }

    // Lock mutex before updating global sum
    pthread_mutex_lock(&mutex);
    sumbuf += local_sum;
    pthread_mutex_unlock(&mutex);

    pthread_exit(NULL);
}

int main(int argc, char* argv[]) {
    if (argc < 3 || argc > 4) {
```

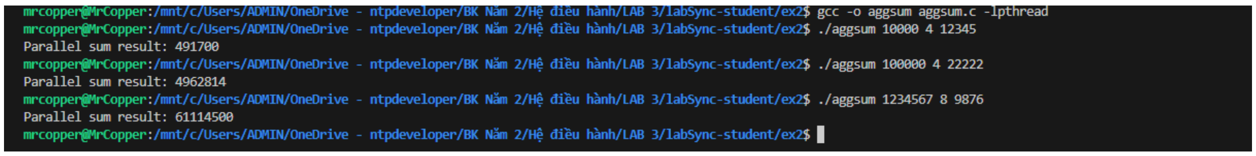


```c
        printf("Usage: %s <arrsz> <tnum> [seednum]\n", argv[0]);
        return 1;
    }

    // Parse arguments
    int arrsz = atoi(argv[1]);
    int tnum = atoi(argv[2]);
    int seednum = (argc == 4) ? atoi(argv[3]) : time(NULL);

    if (arrsz < tnum || tnum <= 0 || arrsz <= 0) {
        printf("Invalid arguments: array size must be >= thread count, both > 0\n");
        return 1;
    }

    // Allocate array
    shrdarrbuf = (int*)malloc(arrsz * sizeof(int));
    if (!shrdarrbuf) {
        perror("Memory allocation failed");
        return 1;
    }

    // Generate random array data
    generate_array_data(shrdarrbuf, arrsz, seednum);

    // Initialize mutex
    pthread_mutex_init(&mutex, NULL);

    // Create threads
    pthread_t threads[tnum];
    struct range thread_ranges[tnum];
    int chunk_size = arrsz / tnum;
    int remainder = arrsz % tnum;

    int start = 0;
    for (int i = 0; i < tnum; i++) {
        thread_ranges[i].start = start;
        thread_ranges[i].end = start + chunk_size - 1;
        if (remainder > 0) {
            thread_ranges[i].end++;
            remainder--;
        }
        start = thread_ranges[i].end + 1;

        pthread_create(&threads[i], NULL, sum_worker, &thread_ranges[i]);
    }

    // Join threads
    for (int i = 0; i < tnum; i++) {
        pthread_join(threads[i], NULL);
    }
```



```c

    // Destroy mutex and free memory
    pthread_mutex_destroy(&mutex);
    free(shrdarrbuf);

    // Print results
    printf("Parallel sum result: %ld\n", sumbuf);

    return 0;
}

```

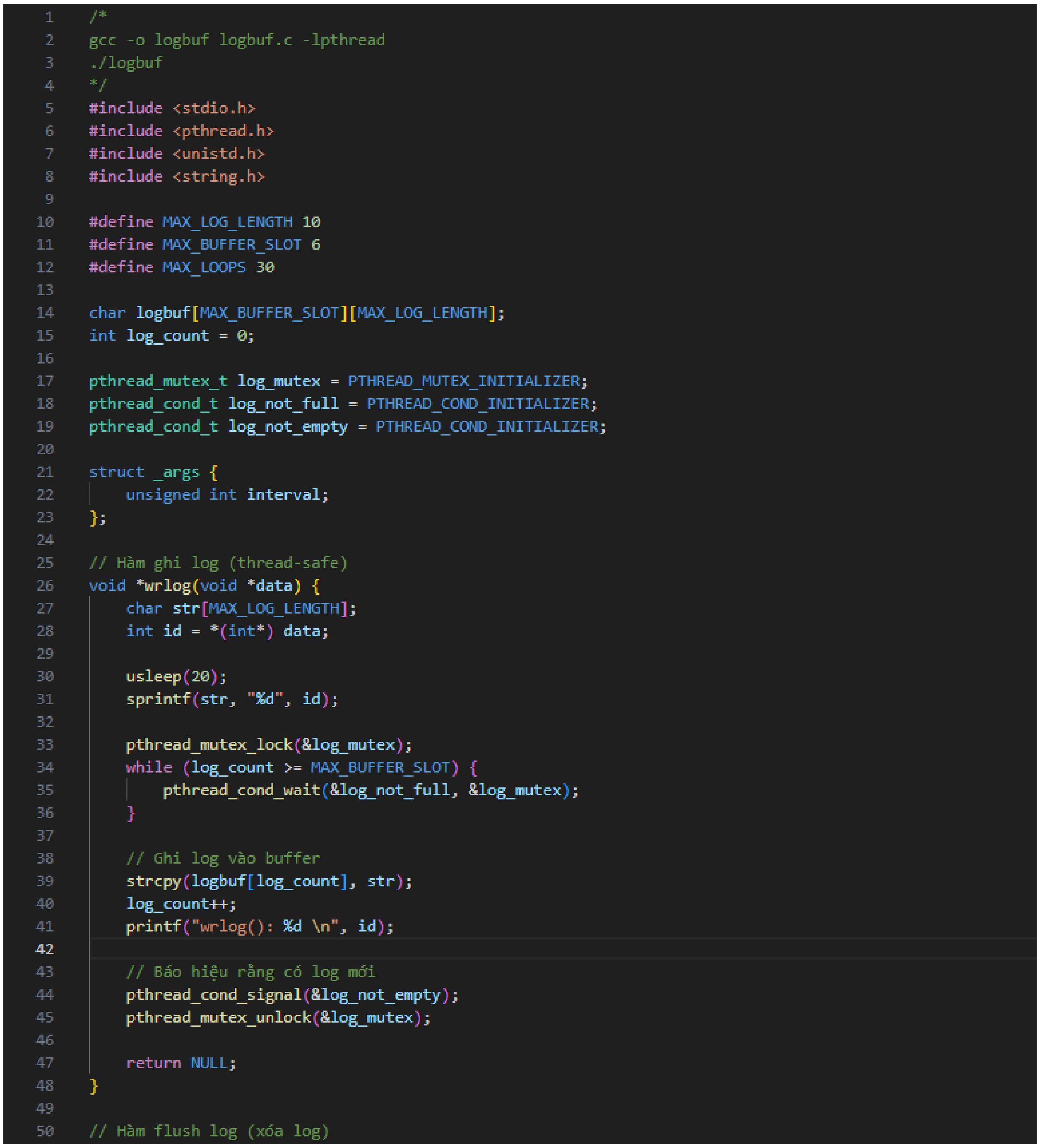
Output:



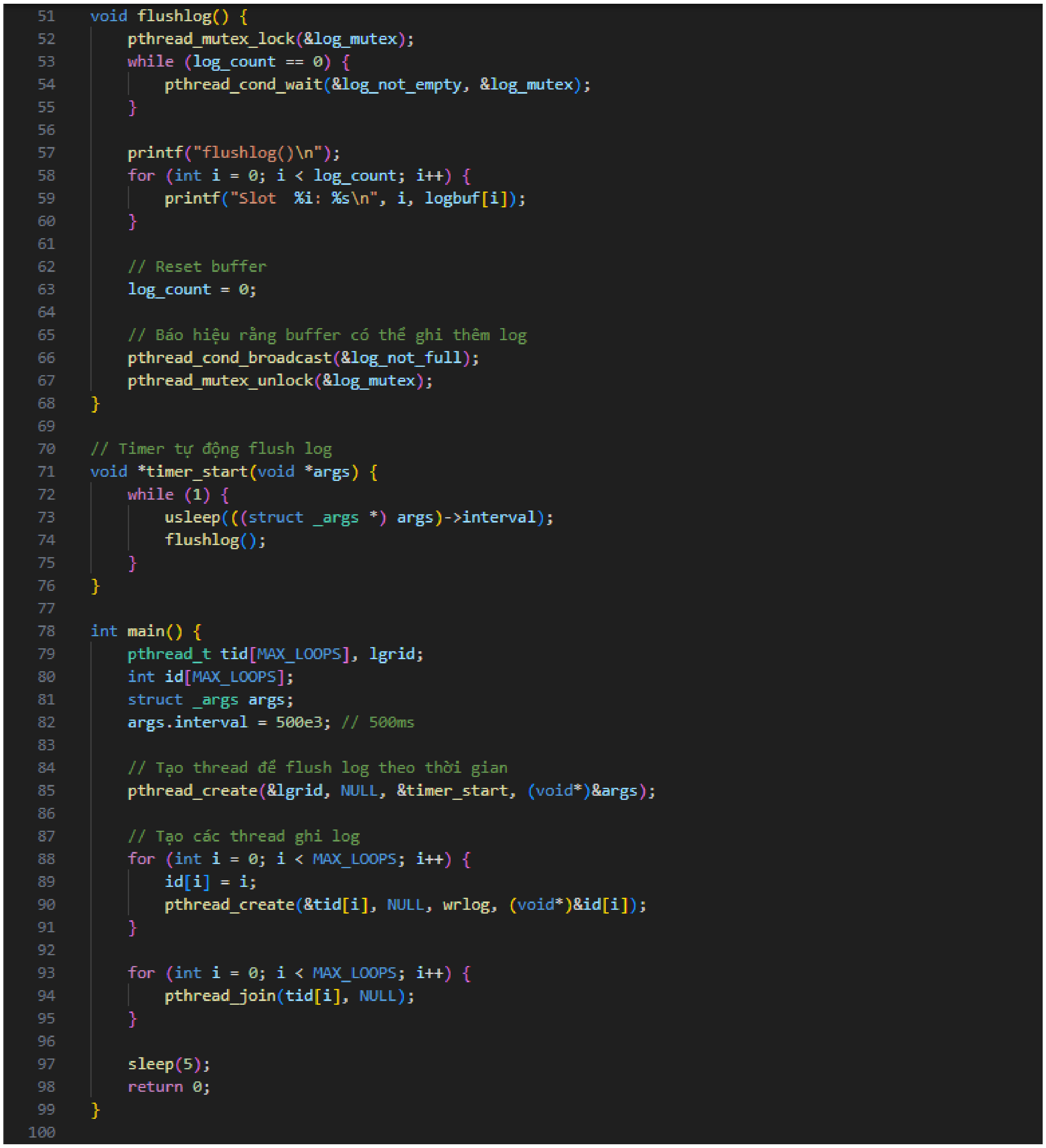
```
mrcopper@MrCopper:/mnt/c/Users/ADMIN/OneDrive - ntpdeveloper/BK Năm 2/Hệ điều hành/LAB 3/labSync-student/ex2$ gcc -o aggsum aggsum.c -lpthread
mrcopper@MrCopper:/mnt/c/Users/ADMIN/OneDrive - ntpdeveloper/BK Năm 2/Hệ điều hành/LAB 3/labSync-student/ex2$ ./aggsum 10000 4 12345
Parallel sum result: 491700
mrcopper@MrCopper:/mnt/c/Users/ADMIN/OneDrive - ntpdeveloper/BK Năm 2/Hệ điều hành/LAB 3/labSync-student/ex2$ ./aggsum 100000 4 22222
Parallel sum result: 4962814
mrcopper@MrCopper:/mnt/c/Users/ADMIN/OneDrive - ntpdeveloper/BK Năm 2/Hệ điều hành/LAB 3/labSync-student/ex2$ ./aggsum 1234567 8 9876
Parallel sum result: 61114500
mrcopper@MrCopper:/mnt/c/Users/ADMIN/OneDrive - ntpdeveloper/BK Năm 2/Hệ điều hành/LAB 3/labSync-student/ex2$
```

PROBLEM 3

Code:

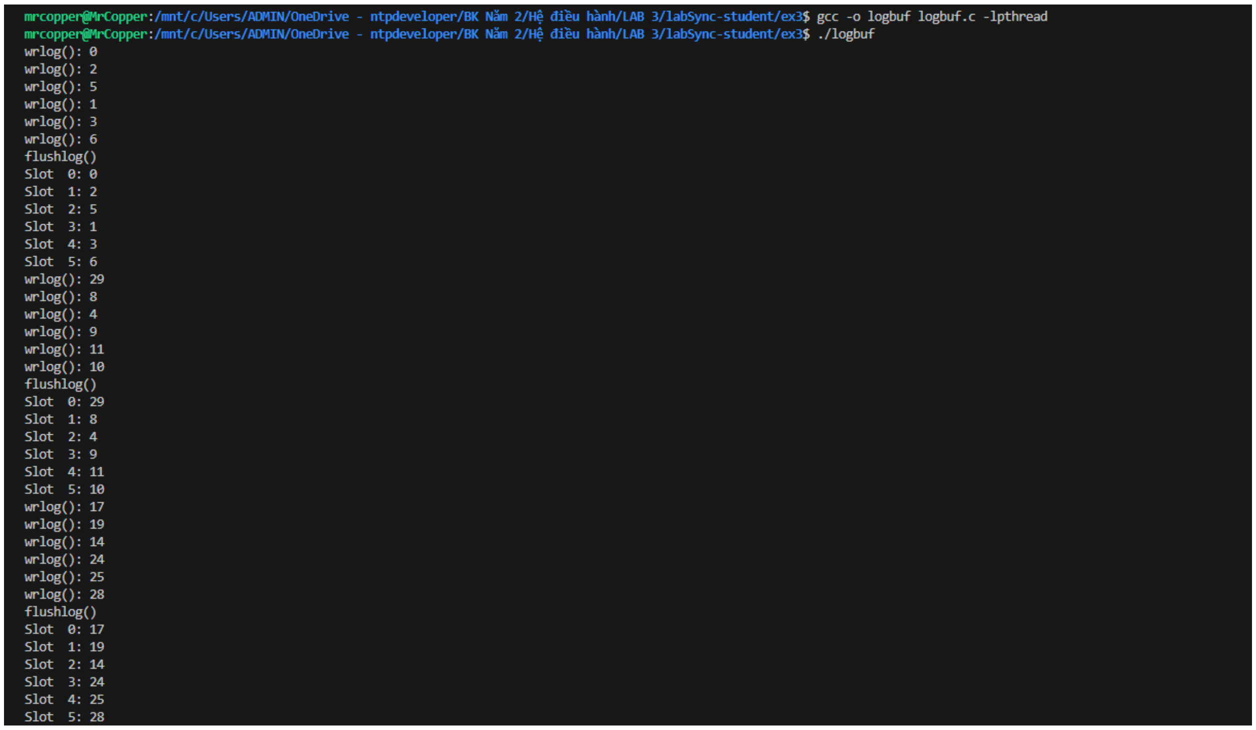


```c
/*
gcc -o logbuf logbuf.c -lpthread
./logbuf
*/
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>
#include <string.h>

#define MAX_LOG_LENGTH 10
#define MAX_BUFFER_SLOT 6
#define MAX_LOOPS 30

char logbuf[MAX_BUFFER_SLOT][MAX_LOG_LENGTH];
int log_count = 0;

pthread_mutex_t log_mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t log_not_full = PTHREAD_COND_INITIALIZER;
pthread_cond_t log_not_empty = PTHREAD_COND_INITIALIZER;

struct _args {
    unsigned int interval;
};

// Hàm ghi log (thread-safe)
void *wrlog(void *data) {
    char str[MAX_LOG_LENGTH];
    int id = *(int*) data;

    usleep(20);
    sprintf(str, "%d", id);

    pthread_mutex_lock(&log_mutex);
    while (log_count >= MAX_BUFFER_SLOT) {
        pthread_cond_wait(&log_not_full, &log_mutex);
    }

    // Ghi log vào buffer
    strcpy(logbuf[log_count], str);
    log_count++;
    printf("wrlog(): %d \n", id);

    // Báo hiệu rằng có log mới
    pthread_cond_signal(&log_not_empty);
    pthread_mutex_unlock(&log_mutex);

    return NULL;
}

// Hàm flush log (xóa log)
```

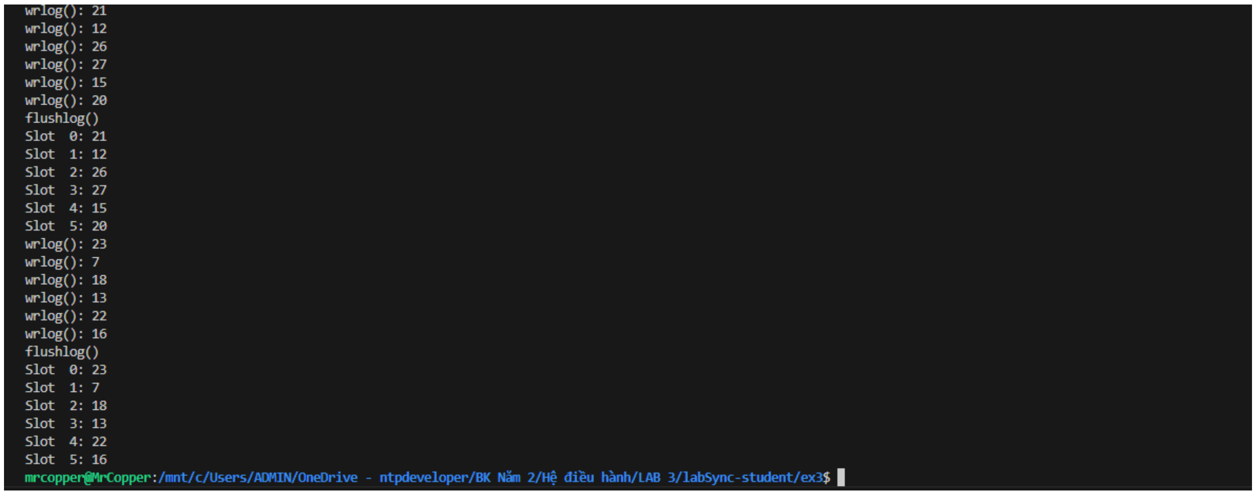


```c
void flushlog() {
    pthread_mutex_lock(&log_mutex);
    while (log_count == 0) {
        pthread_cond_wait(&log_not_empty, &log_mutex);
    }

    printf("flushlog()\n");
    for (int i = 0; i < log_count; i++) {
        printf("Slot  %i: %s\n", i, logbuf[i]);
    }

    // Reset buffer
    log_count = 0;

    // Báo hiệu rằng buffer có thể ghi thêm log
    pthread_cond_broadcast(&log_not_full);
    pthread_mutex_unlock(&log_mutex);
}

// Timer tự động flush log
void *timer_start(void *args) {
    while (1) {
        usleep(((struct _args *) args)->interval);
        flushlog();
    }
}

int main() {
    pthread_t tid[MAX_LOOPS], lgrid;
    int id[MAX_LOOPS];
    struct _args args;
    args.interval = 500e3; // 500ms

    // Tạo thread để flush log theo thời gian
    pthread_create(&lgrid, NULL, &timer_start, (void*)&args);

    // Tạo các thread ghi log
    for (int i = 0; i < MAX_LOOPS; i++) {
        id[i] = i;
        pthread_create(&tid[i], NULL, wrlog, (void*)&id[i]);
    }

    for (int i = 0; i < MAX_LOOPS; i++) {
        pthread_join(tid[i], NULL);
    }

    sleep(5);
    return 0;
}
```

Output:



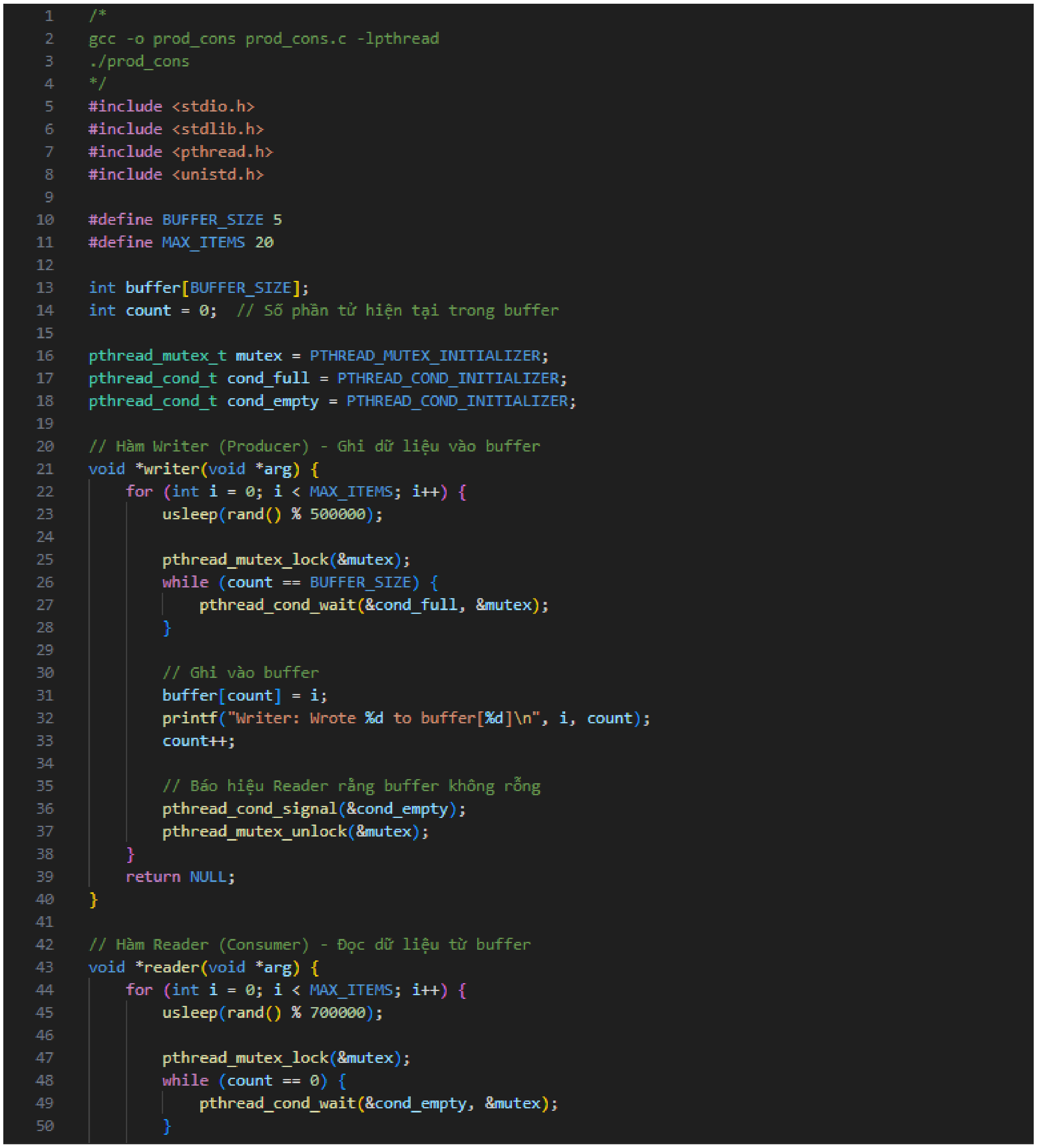
```
mrcopper@MrCopper:/mnt/c/Users/ADMIN/OneDrive - ntpdeveloper/BK Năm 2/Hệ điều hành/LAB 3/labSync-student/ex3$ gcc -o logbuf logbuf.c -lpthread
mrcopper@MrCopper:/mnt/c/Users/ADMIN/OneDrive - ntpdeveloper/BK Năm 2/Hệ điều hành/LAB 3/labSync-student/ex3$ ./logbuf
wrlog(): 0
wrlog(): 2
wrlog(): 5
wrlog(): 1
wrlog(): 3
wrlog(): 6
flushlog()
Slot  0: 0
Slot  1: 2
Slot  2: 5
Slot  3: 1
Slot  4: 3
Slot  5: 6
wrlog(): 29
wrlog(): 8
wrlog(): 4
wrlog(): 9
wrlog(): 11
wrlog(): 10
flushlog()
Slot  0: 29
Slot  1: 8
Slot  2: 4
Slot  3: 9
Slot  4: 11
Slot  5: 10
wrlog(): 17
wrlog(): 19
wrlog(): 14
wrlog(): 24
wrlog(): 25
wrlog(): 28
flushlog()
Slot  0: 17
Slot  1: 19
Slot  2: 14
Slot  3: 24
Slot  4: 25
Slot  5: 28
```



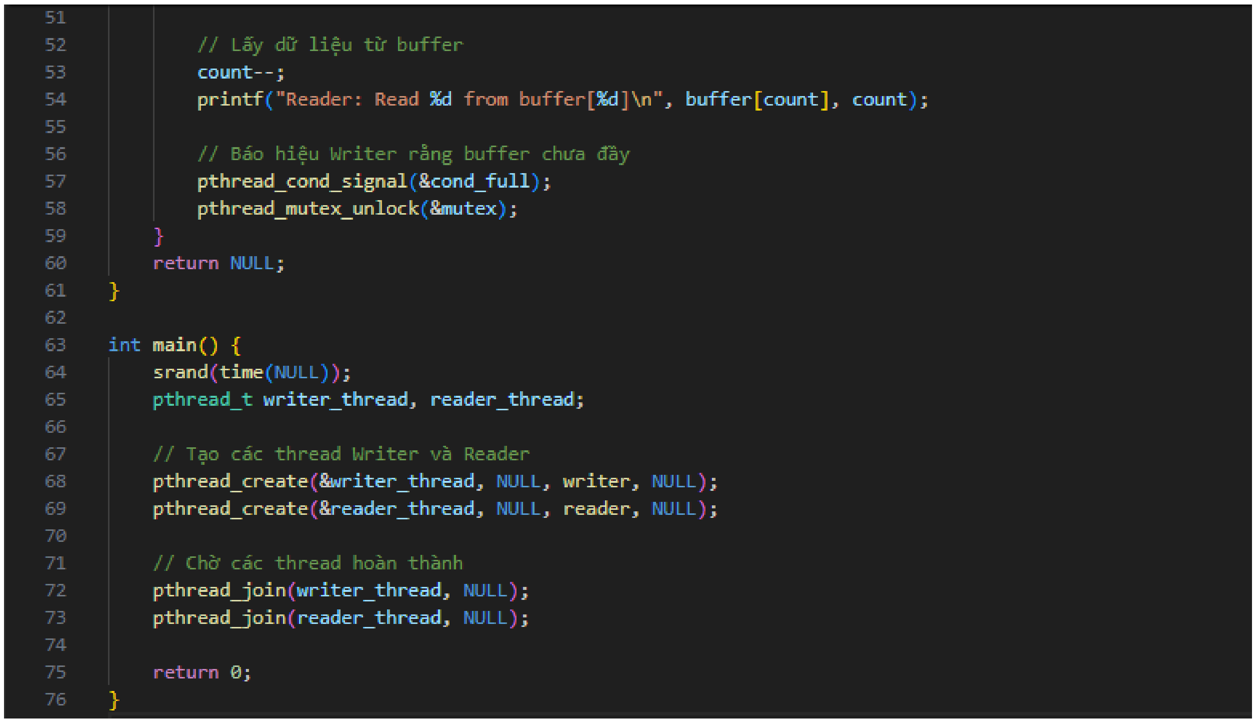
```
wrlog(): 21
wrlog(): 12
wrlog(): 26
wrlog(): 27
wrlog(): 15
wrlog(): 20
flushlog()
Slot  0: 21
Slot  1: 12
Slot  2: 26
Slot  3: 27
Slot  4: 15
Slot  5: 20
wrlog(): 23
wrlog(): 7
wrlog(): 18
wrlog(): 13
wrlog(): 22
wrlog(): 16
flushlog()
Slot  0: 23
Slot  1: 7
Slot  2: 18
Slot  3: 13
Slot  4: 22
Slot  5: 16
mrcopper@MrCopper:/mnt/c/Users/ADMIN/OneDrive - ntpdeveloper/BK Năm 2/Hệ điều hành/LAB 3/labSync-student/ex3$
```

PROBLEM 4

Code:

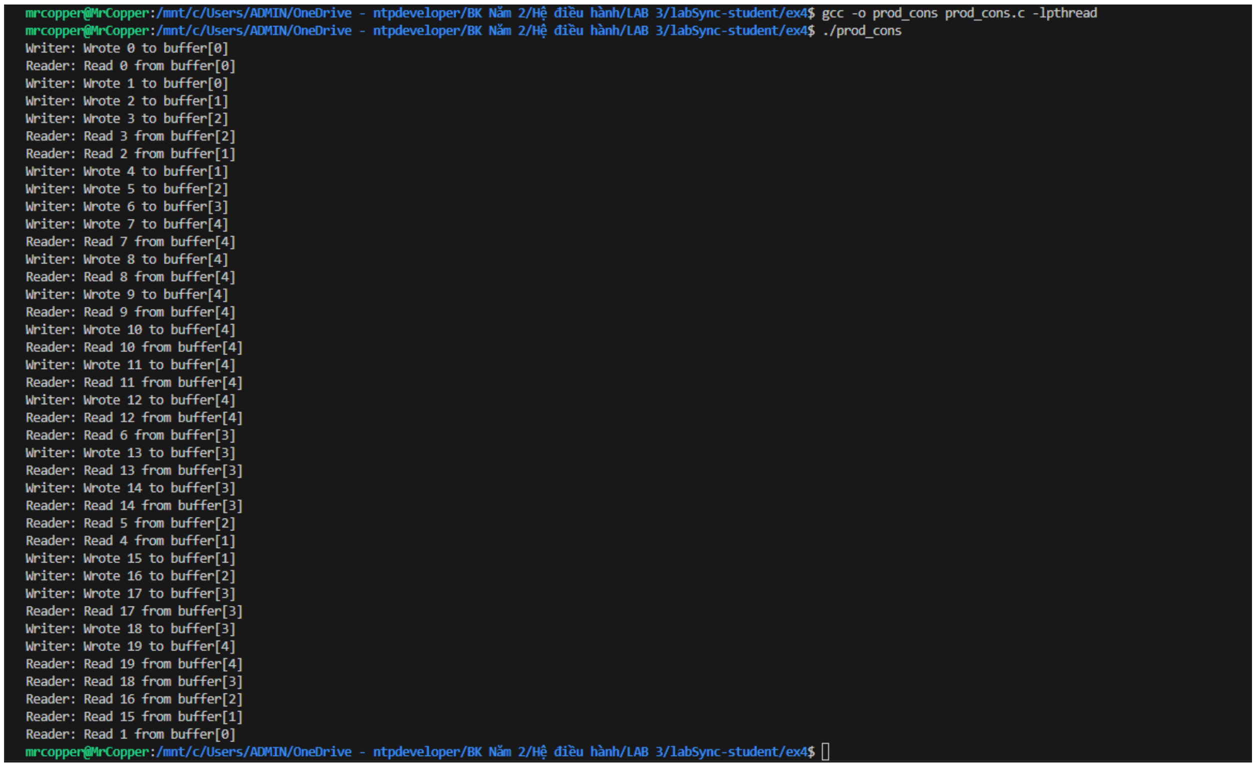


```c
/*
gcc -o prod_cons prod_cons.c -lpthread
./prod_cons
*/
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>

#define BUFFER_SIZE 5
#define MAX_ITEMS 20

int buffer[BUFFER_SIZE];
int count = 0;  // Số phần tử hiện tại trong buffer

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t cond_full = PTHREAD_COND_INITIALIZER;
pthread_cond_t cond_empty = PTHREAD_COND_INITIALIZER;

// Hàm Writer (Producer) - Ghi dữ liệu vào buffer
void *writer(void *arg) {
    for (int i = 0; i < MAX_ITEMS; i++) {
        usleep(rand() % 500000);

        pthread_mutex_lock(&mutex);
        while (count == BUFFER_SIZE) {
            pthread_cond_wait(&cond_full, &mutex);
        }

        // Ghi vào buffer
        buffer[count] = i;
        printf("Writer: Wrote %d to buffer[%d]\n", i, count);
        count++;

        // Báo hiệu Reader rằng buffer không rỗng
        pthread_cond_signal(&cond_empty);
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}

// Hàm Reader (Consumer) - Đọc dữ liệu từ buffer
void *reader(void *arg) {
    for (int i = 0; i < MAX_ITEMS; i++) {
        usleep(rand() % 700000);

        pthread_mutex_lock(&mutex);
        while (count == 0) {
            pthread_cond_wait(&cond_empty, &mutex);
        }
```



```c

        // Lấy dữ liệu từ buffer
        count--;
        printf("Reader: Read %d from buffer[%d]\n", buffer[count], count);

        // Báo hiệu Writer rằng buffer chưa đầy
        pthread_cond_signal(&cond_full);
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}

int main() {
    srand(time(NULL));
    pthread_t writer_thread, reader_thread;

    // Tạo các thread Writer và Reader
    pthread_create(&writer_thread, NULL, writer, NULL);
    pthread_create(&reader_thread, NULL, reader, NULL);

    // Chờ các thread hoàn thành
    pthread_join(writer_thread, NULL);
    pthread_join(reader_thread, NULL);

    return 0;
}
```

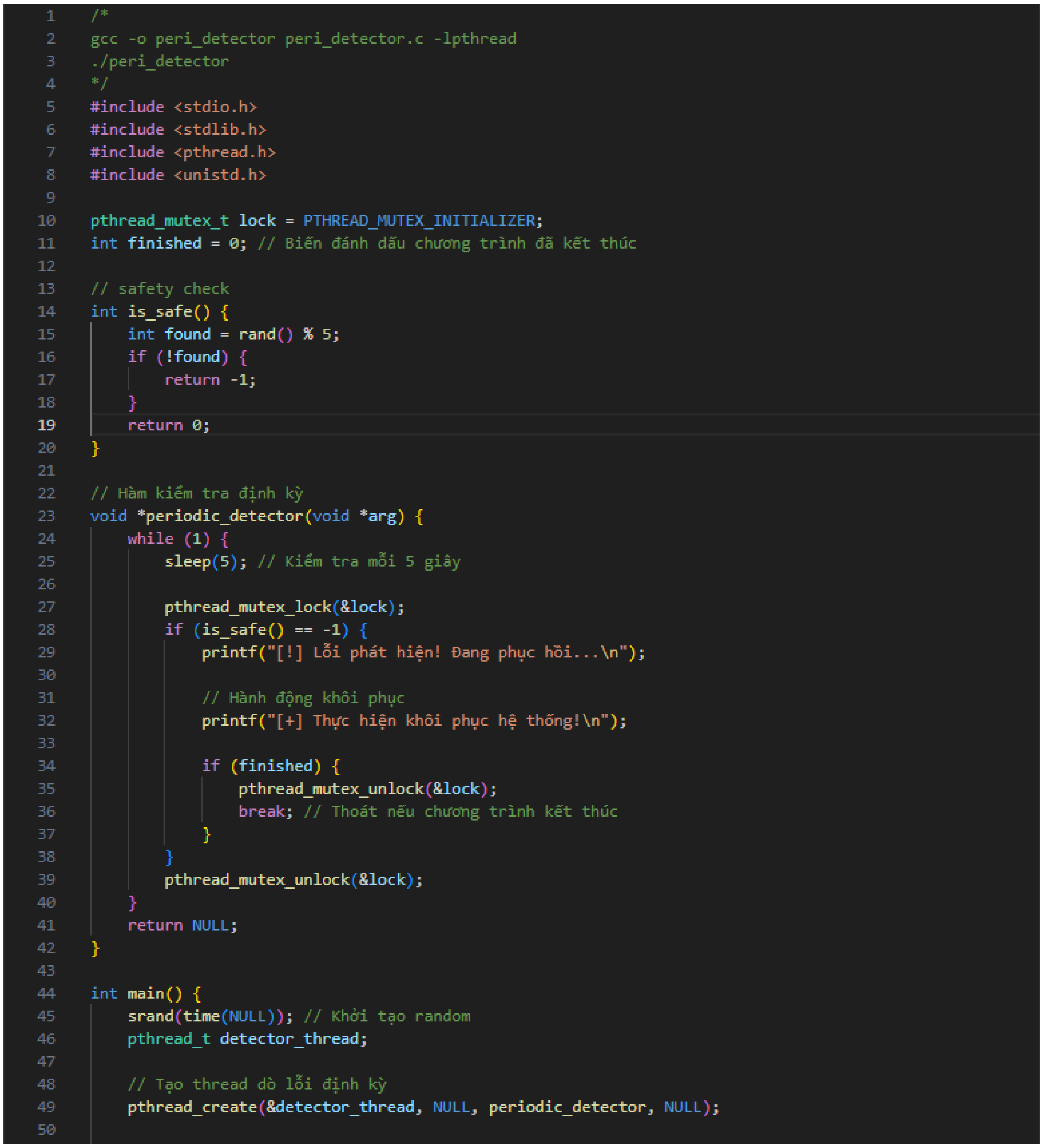
Output:



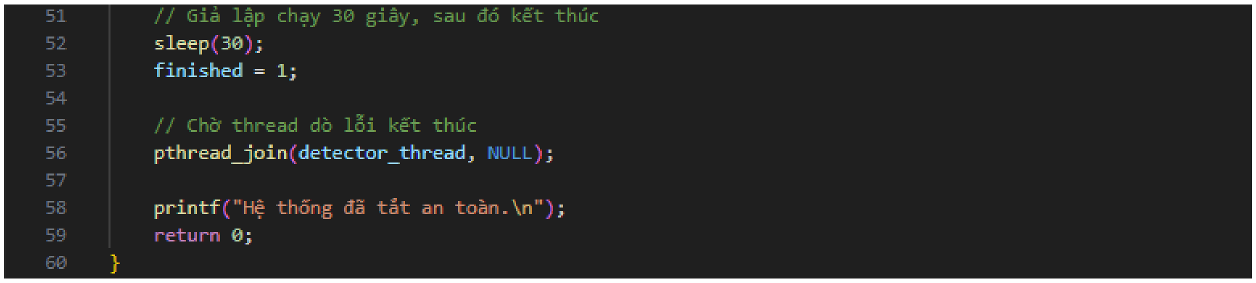
```
mrcopper@MrCopper:/mnt/c/Users/ADMIN/OneDrive - ntpdeveloper/BK Năm 2/Hệ điều hành/LAB 3/labSync-student/ex4$ gcc -o prod_cons prod_cons.c -lpthread
mrcopper@MrCopper:/mnt/c/Users/ADMIN/OneDrive - ntpdeveloper/BK Năm 2/Hệ điều hành/LAB 3/labSync-student/ex4$ ./prod_cons
Writer: Wrote 0 to buffer[0]
Reader: Read 0 from buffer[0]
Writer: Wrote 1 to buffer[0]
Writer: Wrote 2 to buffer[1]
Writer: Wrote 3 to buffer[2]
Reader: Read 3 from buffer[2]
Reader: Read 2 from buffer[1]
Writer: Wrote 4 to buffer[1]
Writer: Wrote 5 to buffer[2]
Writer: Wrote 6 to buffer[3]
Writer: Wrote 7 to buffer[4]
Reader: Read 7 from buffer[4]
Writer: Wrote 8 to buffer[4]
Reader: Read 8 from buffer[4]
Writer: Wrote 9 to buffer[4]
Reader: Read 9 from buffer[4]
Writer: Wrote 10 to buffer[4]
Reader: Read 10 from buffer[4]
Writer: Wrote 11 to buffer[4]
Reader: Read 11 from buffer[4]
Writer: Wrote 12 to buffer[4]
Reader: Read 12 from buffer[4]
Reader: Read 6 from buffer[3]
Writer: Wrote 13 to buffer[3]
Reader: Read 13 from buffer[3]
Writer: Wrote 14 to buffer[3]
Reader: Read 14 from buffer[3]
Reader: Read 5 from buffer[2]
Reader: Read 4 from buffer[1]
Writer: Wrote 15 to buffer[1]
Writer: Wrote 16 to buffer[2]
Writer: Wrote 17 to buffer[3]
Reader: Read 17 from buffer[3]
Writer: Wrote 18 to buffer[3]
Writer: Wrote 19 to buffer[4]
Reader: Read 19 from buffer[4]
Reader: Read 18 from buffer[3]
Reader: Read 16 from buffer[2]
Reader: Read 15 from buffer[1]
Reader: Read 1 from buffer[0]
mrcopper@MrCopper:/mnt/c/Users/ADMIN/OneDrive - ntpdeveloper/BK Năm 2/Hệ điều hành/LAB 3/labSync-student/ex4$ 
```

PROBLEM 5

Code:

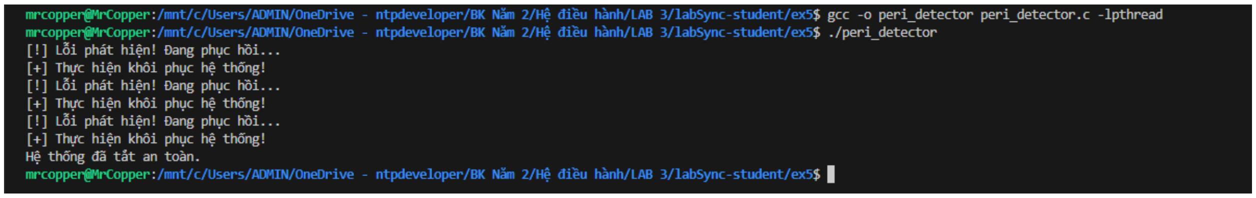


```c
/*
gcc -o peri_detector peri_detector.c -lpthread
./peri_detector
*/
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>

pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;
int finished = 0; // Biến đánh dấu chương trình đã kết thúc

// safety check
int is_safe() {
    int found = rand() % 5;
    if (!found) {
        return -1;
    }
    return 0;
}

// Hàm kiểm tra định kỳ
void *periodic_detector(void *arg) {
    while (1) {
        sleep(5); // Kiểm tra mỗi 5 giây

        pthread_mutex_lock(&lock);
        if (is_safe() == -1) {
            printf("[!] Lỗi phát hiện! Đang phục hồi...\n");

            // Hành động khôi phục
            printf("[+] Thực hiện khôi phục hệ thống!\n");

            if (finished) {
                pthread_mutex_unlock(&lock);
                break; // Thoát nếu chương trình kết thúc
            }
        }
        pthread_mutex_unlock(&lock);
    }
    return NULL;
}

int main() {
    srand(time(NULL)); // Khởi tạo random
    pthread_t detector_thread;

    // Tạo thread dò lỗi định kỳ
    pthread_create(&detector_thread, NULL, periodic_detector, NULL);

```



```c
    // Giả lập chạy 30 giây, sau đó kết thúc
    sleep(30);
    finished = 1;

    // Chờ thread dò lỗi kết thúc
    pthread_join(detector_thread, NULL);

    printf("Hệ thống đã tắt an toàn.\n");
    return 0;
}
```

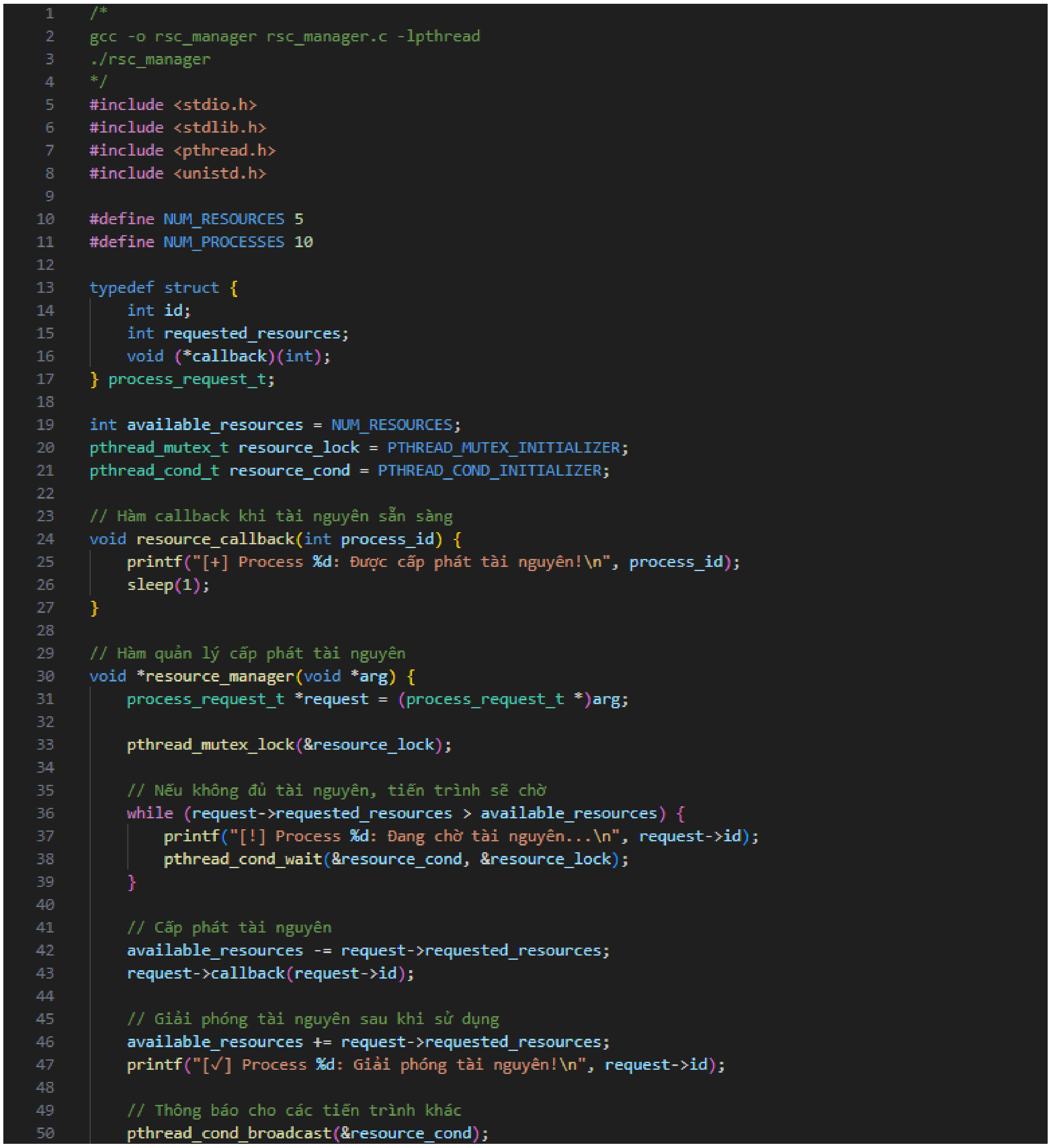
Output:



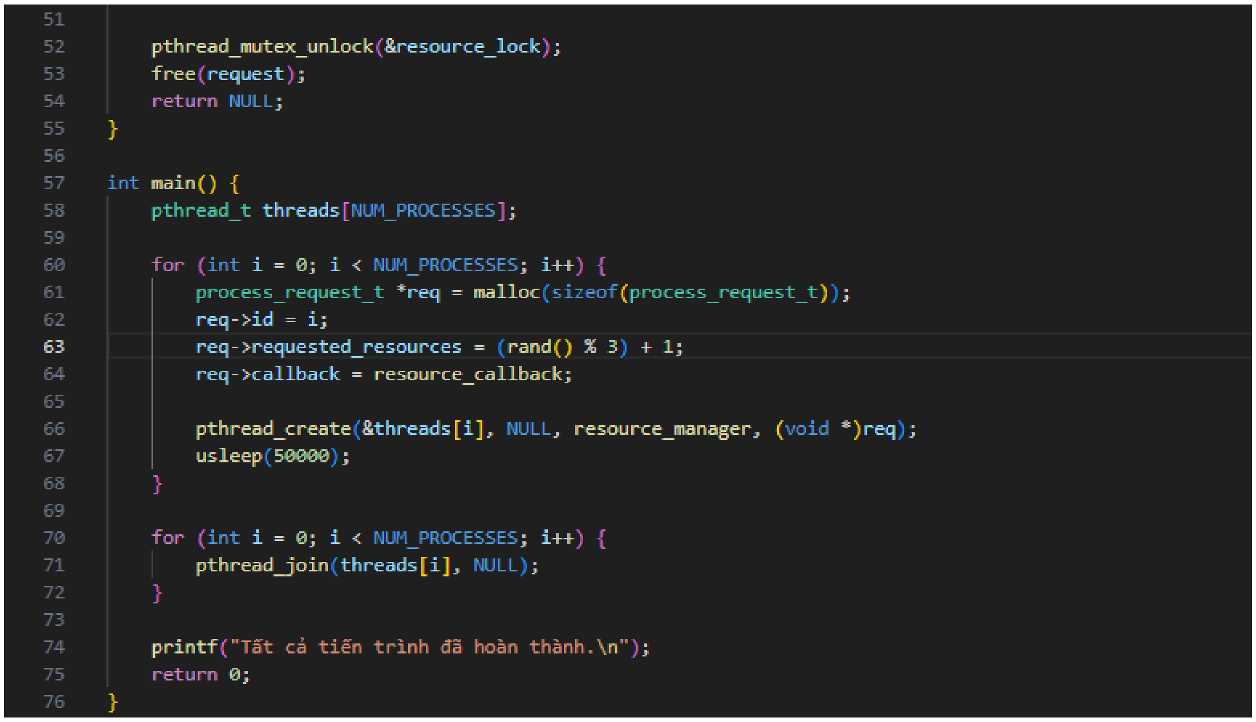
```
mrcopper@MrCopper:/mnt/c/Users/ADMIN/OneDrive - ntpdeveloper/BK Năm 2/Hệ điều hành/LAB 3/labSync-student/ex5$ gcc -o peri_detector peri_detector.c -lpthread
mrcopper@MrCopper:/mnt/c/Users/ADMIN/OneDrive - ntpdeveloper/BK Năm 2/Hệ điều hành/LAB 3/labSync-student/ex5$ ./peri_detector
[!] Lỗi phát hiện! Đang phục hồi...
[+] Thực hiện khôi phục hệ thống!
[!] Lỗi phát hiện! Đang phục hồi...
[+] Thực hiện khôi phục hệ thống!
[!] Lỗi phát hiện! Đang phục hồi...
[+] Thực hiện khôi phục hệ thống!
Hệ thống đã tắt an toàn.
mrcopper@MrCopper:/mnt/c/Users/ADMIN/OneDrive - ntpdeveloper/BK Năm 2/Hệ điều hành/LAB 3/labSync-student/ex5$
```

PROBLEM 6

Code:

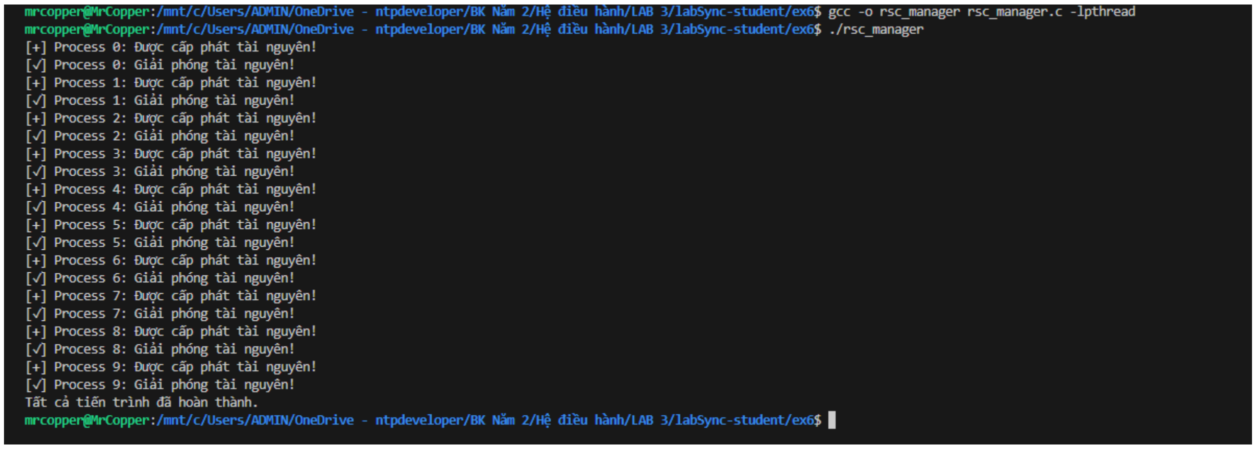


```c
/*
gcc -o rsc_manager rsc_manager.c -lpthread
./rsc_manager
*/
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>

#define NUM_RESOURCES 5
#define NUM_PROCESSES 10

typedef struct {
    int id;
    int requested_resources;
    void (*callback)(int);
} process_request_t;

int available_resources = NUM_RESOURCES;
pthread_mutex_t resource_lock = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t resource_cond = PTHREAD_COND_INITIALIZER;

// Hàm callback khi tài nguyên sẵn sàng
void resource_callback(int process_id) {
    printf("[+] Process %d: Được cấp phát tài nguyên!\n", process_id);
    sleep(1);
}

// Hàm quản lý cấp phát tài nguyên
void *resource_manager(void *arg) {
    process_request_t *request = (process_request_t *)arg;

    pthread_mutex_lock(&resource_lock);

    // Nếu không đủ tài nguyên, tiến trình sẽ chờ
    while (request->requested_resources > available_resources) {
        printf("[!] Process %d: Đang chờ tài nguyên...\n", request->id);
        pthread_cond_wait(&resource_cond, &resource_lock);
    }

    // Cấp phát tài nguyên
    available_resources -= request->requested_resources;
    request->callback(request->id);

    // Giải phóng tài nguyên sau khi sử dụng
    available_resources += request->requested_resources;
    printf("[√] Process %d: Giải phóng tài nguyên!\n", request->id);

    // Thông báo cho các tiến trình khác
    pthread_cond_broadcast(&resource_cond);
```



```c

    pthread_mutex_unlock(&resource_lock);
    free(request);
    return NULL;
}

int main() {
    pthread_t threads[NUM_PROCESSES];

    for (int i = 0; i < NUM_PROCESSES; i++) {
        process_request_t *req = malloc(sizeof(process_request_t));
        req->id = i;
        req->requested_resources = (rand() % 3) + 1;
        req->callback = resource_callback;

        pthread_create(&threads[i], NULL, resource_manager, (void *)req);
        usleep(50000);
    }

    for (int i = 0; i < NUM_PROCESSES; i++) {
        pthread_join(threads[i], NULL);
    }

    printf("Tất cả tiến trình đã hoàn thành.\n");
    return 0;
}
```

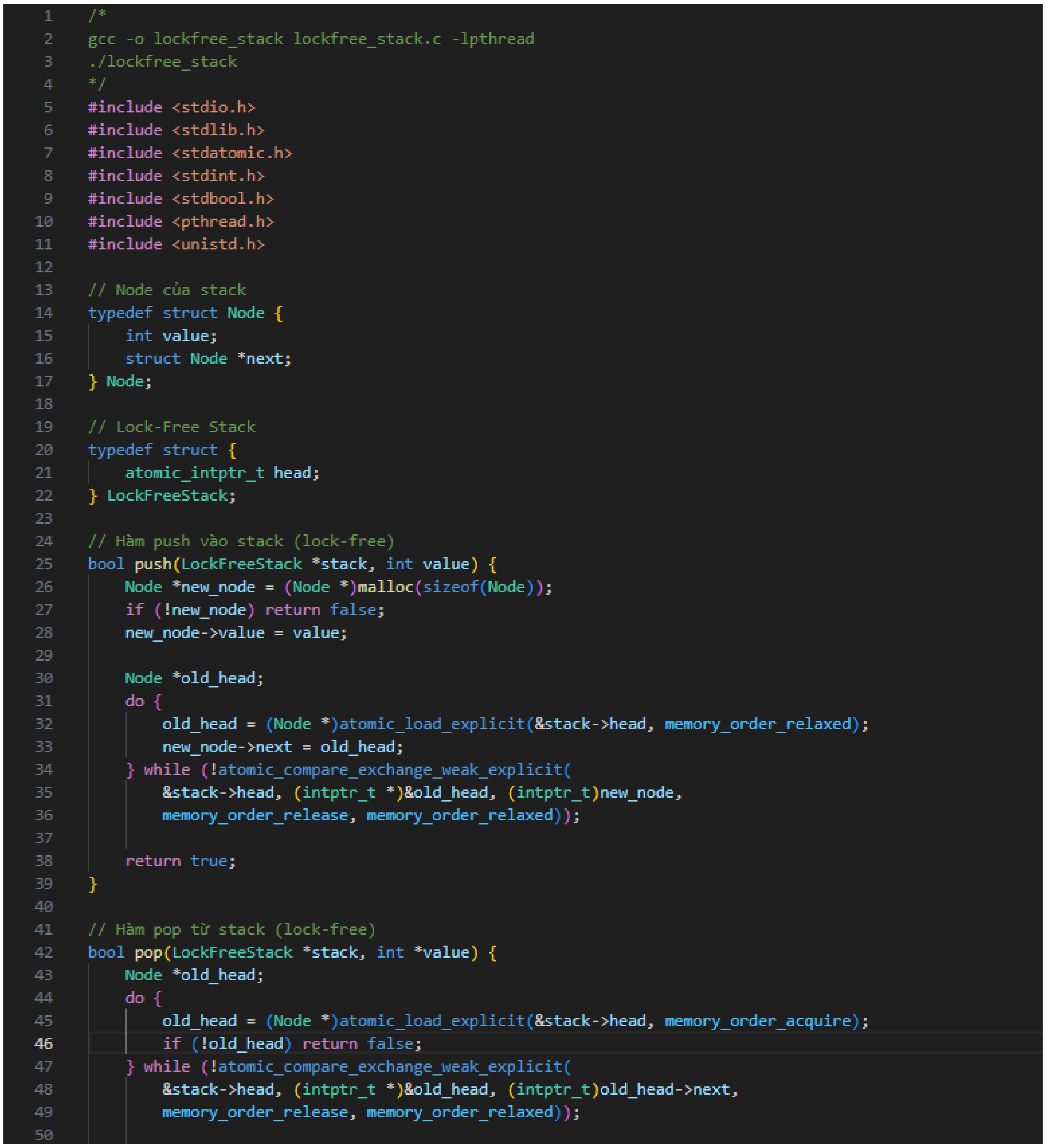
Output:



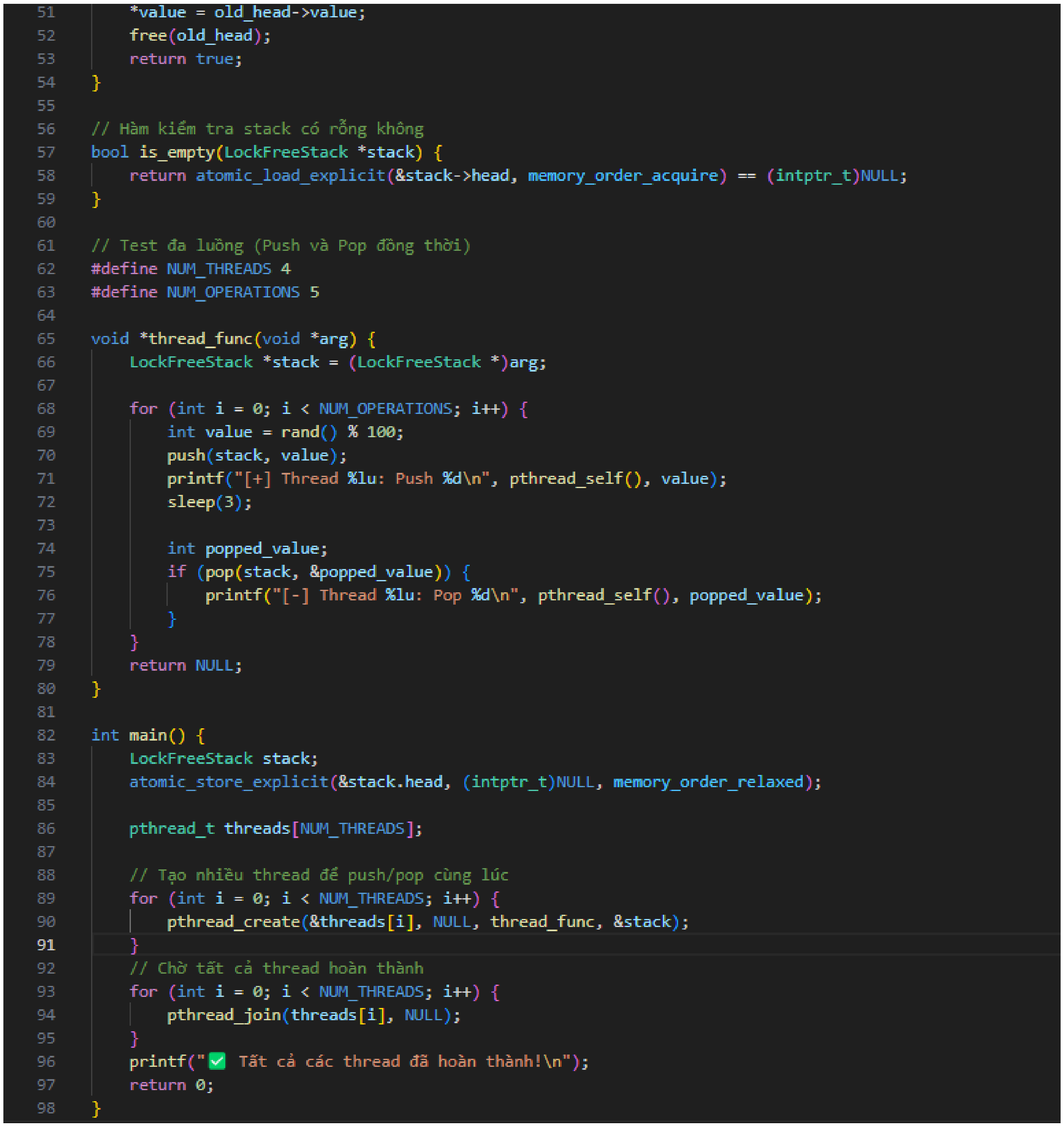
```
mrcopper@MrCopper:/mnt/c/Users/ADMIN/OneDrive - ntpdeveloper/BK Năm 2/Hệ điều hành/LAB 3/labSync-student/ex6$ gcc -o rsc_manager rsc_manager.c -lpthread
mrcopper@MrCopper:/mnt/c/Users/ADMIN/OneDrive - ntpdeveloper/BK Năm 2/Hệ điều hành/LAB 3/labSync-student/ex6$ ./rsc_manager
[+] Process 0: Được cấp phát tài nguyên!
[✓] Process 0: Giải phóng tài nguyên!
[+] Process 1: Được cấp phát tài nguyên!
[✓] Process 1: Giải phóng tài nguyên!
[+] Process 2: Được cấp phát tài nguyên!
[✓] Process 2: Giải phóng tài nguyên!
[+] Process 3: Được cấp phát tài nguyên!
[✓] Process 3: Giải phóng tài nguyên!
[+] Process 4: Được cấp phát tài nguyên!
[✓] Process 4: Giải phóng tài nguyên!
[+] Process 5: Được cấp phát tài nguyên!
[✓] Process 5: Giải phóng tài nguyên!
[+] Process 6: Được cấp phát tài nguyên!
[✓] Process 6: Giải phóng tài nguyên!
[+] Process 7: Được cấp phát tài nguyên!
[✓] Process 7: Giải phóng tài nguyên!
[+] Process 8: Được cấp phát tài nguyên!
[✓] Process 8: Giải phóng tài nguyên!
[+] Process 9: Được cấp phát tài nguyên!
[✓] Process 9: Giải phóng tài nguyên!
Tất cả tiến trình đã hoàn thành.
mrcopper@MrCopper:/mnt/c/Users/ADMIN/OneDrive - ntpdeveloper/BK Năm 2/Hệ điều hành/LAB 3/labSync-student/ex6$ 
```

PROBLEM 7

Code:

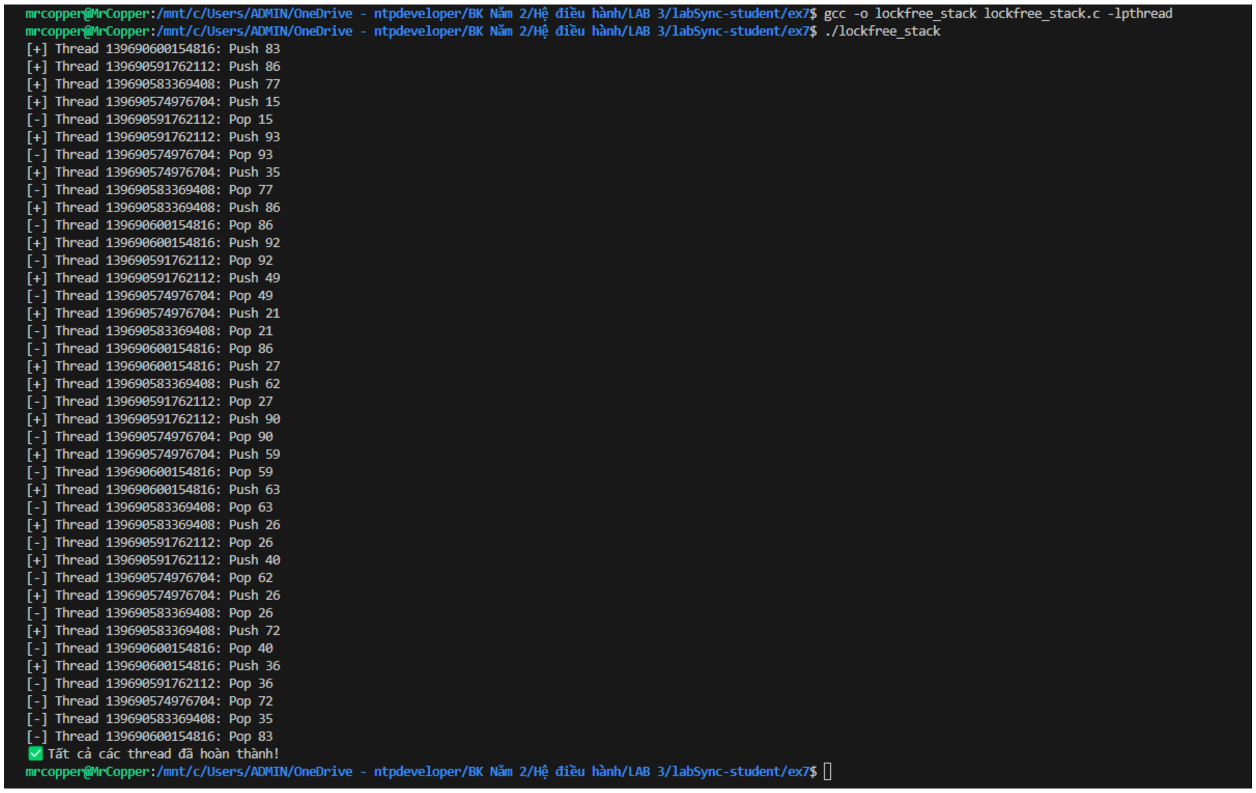


```c
/*
gcc -o lockfree_stack lockfree_stack.c -lpthread
./lockfree_stack
*/
#include <stdio.h>
#include <stdlib.h>
#include <stdatomic.h>
#include <stdint.h>
#include <stdbool.h>
#include <pthread.h>
#include <unistd.h>

// Node của stack
typedef struct Node {
    int value;
    struct Node *next;
} Node;

// Lock-Free Stack
typedef struct {
    atomic_intptr_t head;
} LockFreeStack;

// Hàm push vào stack (lock-free)
bool push(LockFreeStack *stack, int value) {
    Node *new_node = (Node *)malloc(sizeof(Node));
    if (!new_node) return false;
    new_node->value = value;

    Node *old_head;
    do {
        old_head = (Node *)atomic_load_explicit(&stack->head, memory_order_relaxed);
        new_node->next = old_head;
    } while (!atomic_compare_exchange_weak_explicit(
        &stack->head, (intptr_t *)&old_head, (intptr_t)new_node,
        memory_order_release, memory_order_relaxed));

    return true;
}

// Hàm pop từ stack (lock-free)
bool pop(LockFreeStack *stack, int *value) {
    Node *old_head;
    do {
        old_head = (Node *)atomic_load_explicit(&stack->head, memory_order_acquire);
        if (!old_head) return false;
    } while (!atomic_compare_exchange_weak_explicit(
        &stack->head, (intptr_t *)&old_head, (intptr_t)old_head->next,
        memory_order_release, memory_order_relaxed));

```



```c
        *value = old_head->value;
        free(old_head);
        return true;
    }

    // Hàm kiểm tra stack có rỗng không
    bool is_empty(LockFreeStack *stack) {
        return atomic_load_explicit(&stack->head, memory_order_acquire) == (intptr_t)NULL;
    }

    // Test đa luồng (Push và Pop đồng thời)
    #define NUM_THREADS 4
    #define NUM_OPERATIONS 5

    void *thread_func(void *arg) {
        LockFreeStack *stack = (LockFreeStack *)arg;

        for (int i = 0; i < NUM_OPERATIONS; i++) {
            int value = rand() % 100;
            push(stack, value);
            printf("[+] Thread %lu: Push %d\n", pthread_self(), value);
            sleep(3);

            int popped_value;
            if (pop(stack, &popped_value)) {
                printf("[-] Thread %lu: Pop %d\n", pthread_self(), popped_value);
            }
        }
        return NULL;
    }

    int main() {
        LockFreeStack stack;
        atomic_store_explicit(&stack.head, (intptr_t)NULL, memory_order_relaxed);

        pthread_t threads[NUM_THREADS];

        // Tạo nhiều thread để push/pop cùng lúc
        for (int i = 0; i < NUM_THREADS; i++) {
            pthread_create(&threads[i], NULL, thread_func, &stack);
        }
        // Chờ tất cả thread hoàn thành
        for (int i = 0; i < NUM_THREADS; i++) {
            pthread_join(threads[i], NULL);
        }
        printf("✅ Tất cả các thread đã hoàn thành!\n");
        return 0;
    }
```

Output:



```
mrcopper@MrCopper:/mnt/c/Users/ADMIN/OneDrive - ntpdeveloper/BK Năm 2/Hệ điều hành/LAB 3/labSync-student/ex7$ gcc -o lockfree_stack lockfree_stack.c -lpthread
mrcopper@MrCopper:/mnt/c/Users/ADMIN/OneDrive - ntpdeveloper/BK Năm 2/Hệ điều hành/LAB 3/labSync-student/ex7$ ./lockfree_stack
[+] Thread 139690600154816: Push 83
[+] Thread 139690591762112: Push 86
[+] Thread 139690583369408: Push 77
[+] Thread 139690574976704: Push 15
[-] Thread 139690591762112: Pop 15
[+] Thread 139690591762112: Push 93
[-] Thread 139690574976704: Pop 93
[+] Thread 139690574976704: Push 35
[-] Thread 139690583369408: Pop 77
[+] Thread 139690583369408: Push 86
[-] Thread 139690600154816: Pop 86
[+] Thread 139690600154816: Push 92
[-] Thread 139690591762112: Pop 92
[+] Thread 139690591762112: Push 49
[-] Thread 139690574976704: Pop 49
[+] Thread 139690574976704: Push 21
[-] Thread 139690583369408: Pop 21
[-] Thread 139690600154816: Pop 86
[+] Thread 139690600154816: Push 27
[+] Thread 139690583369408: Push 62
[-] Thread 139690591762112: Pop 27
[+] Thread 139690591762112: Push 90
[-] Thread 139690574976704: Pop 90
[+] Thread 139690574976704: Push 59
[-] Thread 139690600154816: Pop 59
[+] Thread 139690600154816: Push 63
[-] Thread 139690583369408: Pop 63
[+] Thread 139690583369408: Push 26
[-] Thread 139690591762112: Pop 26
[+] Thread 139690591762112: Push 40
[-] Thread 139690574976704: Pop 62
[+] Thread 139690574976704: Push 26
[-] Thread 139690583369408: Pop 26
[+] Thread 139690583369408: Push 72
[-] Thread 139690600154816: Pop 40
[+] Thread 139690600154816: Push 36
[-] Thread 139690591762112: Pop 36
[-] Thread 139690574976704: Pop 72
[-] Thread 139690583369408: Pop 35
[-] Thread 139690600154816: Pop 83
✅ Tất cả các thread đã hoàn thành!
mrcopper@MrCopper:/mnt/c/Users/ADMIN/OneDrive - ntpdeveloper/BK Năm 2/Hệ điều hành/LAB 3/labSync-student/ex7$ 
```