CSE344 -- HOMEWORK #4

10/05/2025

Cemal BOLAT

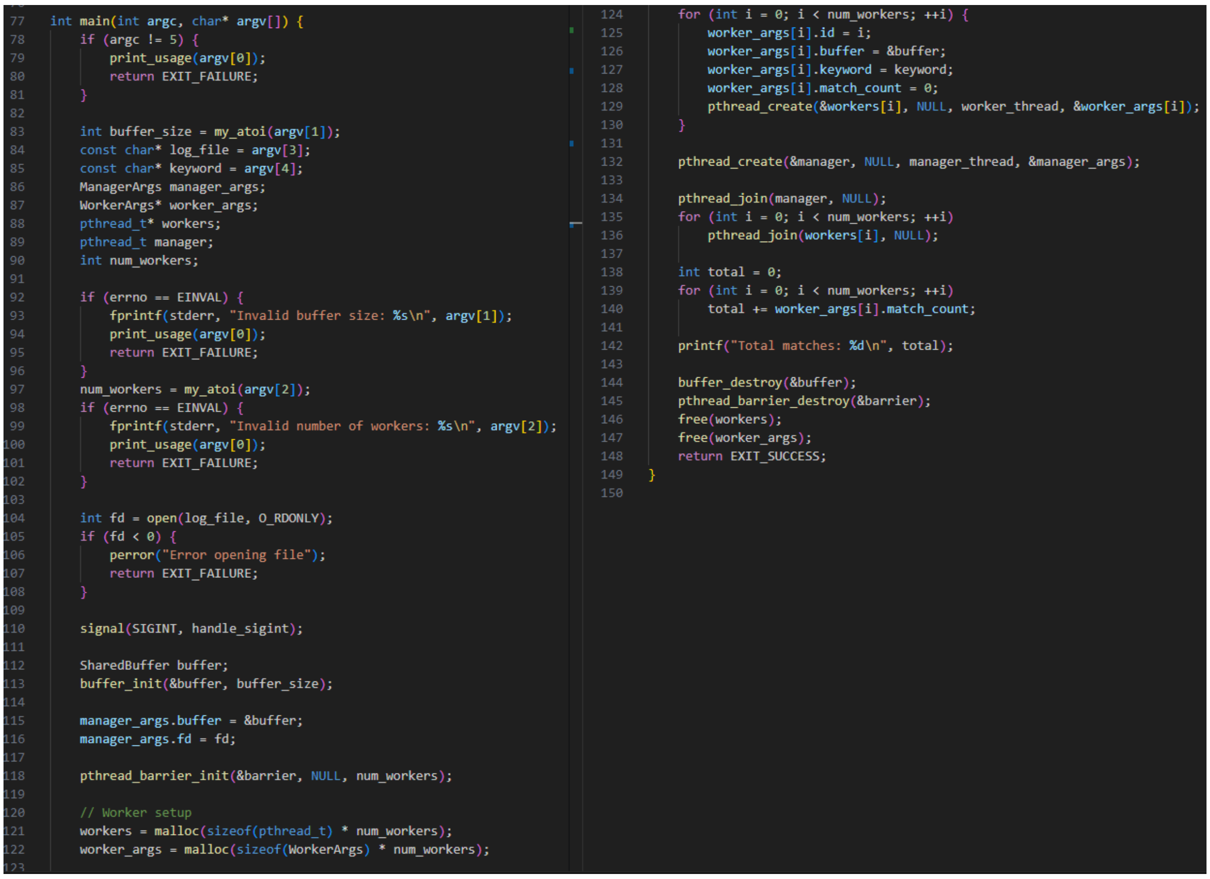
STUDENT NO: 210104004010

Introduction:

This project implements a multithreaded log file analyzer written in C using POSIX threads. The objective is to efficiently search for a specific keyword in a large log file by dividing the workload across multiple worker threads. The program uses a producer-consumer model with synchronization primitives such as mutexes, condition variables, and barriers. The manager thread reads lines from the input file and feeds them to a shared buffer, while worker threads consume these lines, search for the keyword, and report the number of matches they found. Synchronization ensures thread-safe buffer access and coordinated thread termination.

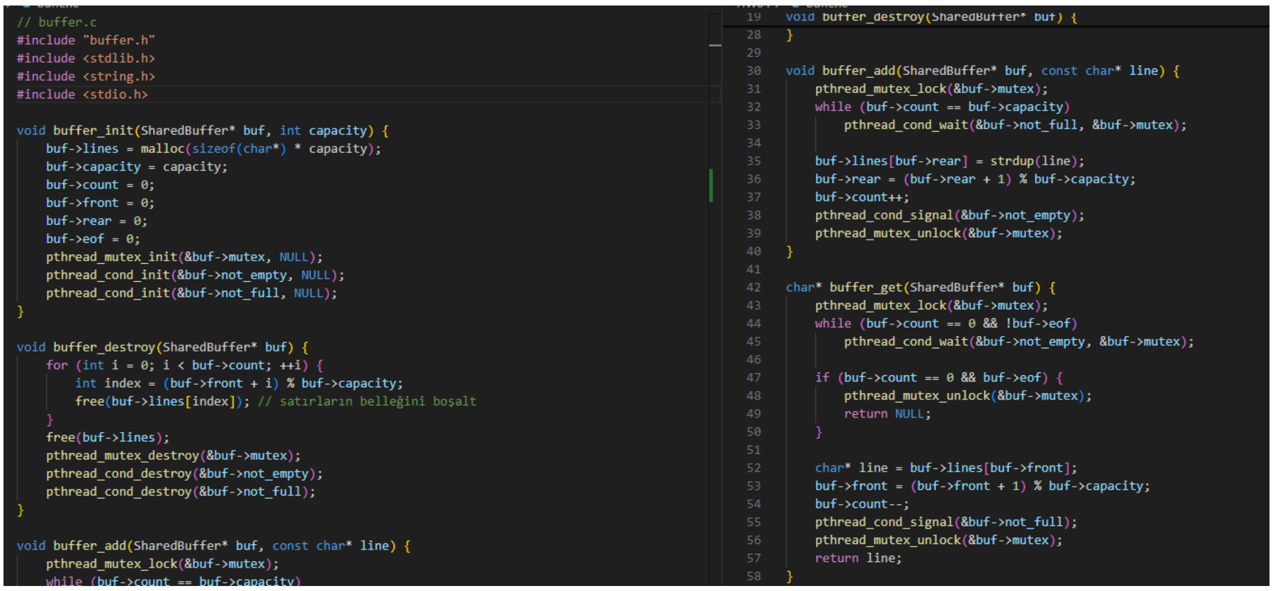
2. Code Explanation

- **main.c**
  - Parses command-line arguments: <buffer_size> <num_workers> <log_file> <search_term>.
  - Opens the log file using open() to get a file descriptor (fd).
  - Initializes the shared buffer, signal handling, and pthread barrier.
  - Launches the manager thread and worker threads.
  - Waits (join) for all threads to complete.
  - Aggregates and prints the final match count.



```c
int main(int argc, char* argv[]) {
    if (argc != 5) {
        print_usage(argv[0]);
        return EXIT_FAILURE;
    }

    int buffer_size = my_atoi(argv[1]);
    const char* log_file = argv[3];
    const char* keyword = argv[4];
    ManagerArgs manager_args;
    WorkerArgs* worker_args;
    pthread_t* workers;
    pthread_t manager;
    int num_workers;

    if (errno == EINVAL) {
        fprintf(stderr, "Invalid buffer size: %s\n", argv[1]);
        print_usage(argv[0]);
        return EXIT_FAILURE;
    }
    num_workers = my_atoi(argv[2]);
    if (errno == EINVAL) {
        fprintf(stderr, "Invalid number of workers: %s\n", argv[2]);
        print_usage(argv[0]);
        return EXIT_FAILURE;
    }

    int fd = open(log_file, O_RDONLY);
    if (fd < 0) {
        perror("Error opening file");
        return EXIT_FAILURE;
    }

    signal(SIGINT, handle_sigint);

    SharedBuffer buffer;
    buffer_init(&buffer, buffer_size);

    manager_args.buffer = &buffer;
    manager_args.fd = fd;

    pthread_barrier_init(&barrier, NULL, num_workers);

    // Worker setup
    workers = malloc(sizeof(pthread_t) * num_workers);
    worker_args = malloc(sizeof(WorkerArgs) * num_workers);

    for (int i = 0; i < num_workers; ++i) {
        worker_args[i].id = i;
        worker_args[i].buffer = &buffer;
        worker_args[i].keyword = keyword;
        worker_args[i].match_count = 0;
        pthread_create(&workers[i], NULL, worker_thread, &worker_args[i]);
    }

    pthread_create(&manager, NULL, manager_thread, &manager_args);

    pthread_join(manager, NULL);
    for (int i = 0; i < num_workers; ++i)
        pthread_join(workers[i], NULL);

    int total = 0;
    for (int i = 0; i < num_workers; ++i)
        total += worker_args[i].match_count;

    printf("Total matches: %d\n", total);

    buffer_destroy(&buffer);
    pthread_barrier_destroy(&barrier);
    free(workers);
    free(worker_args);
    return EXIT_SUCCESS;
}
```

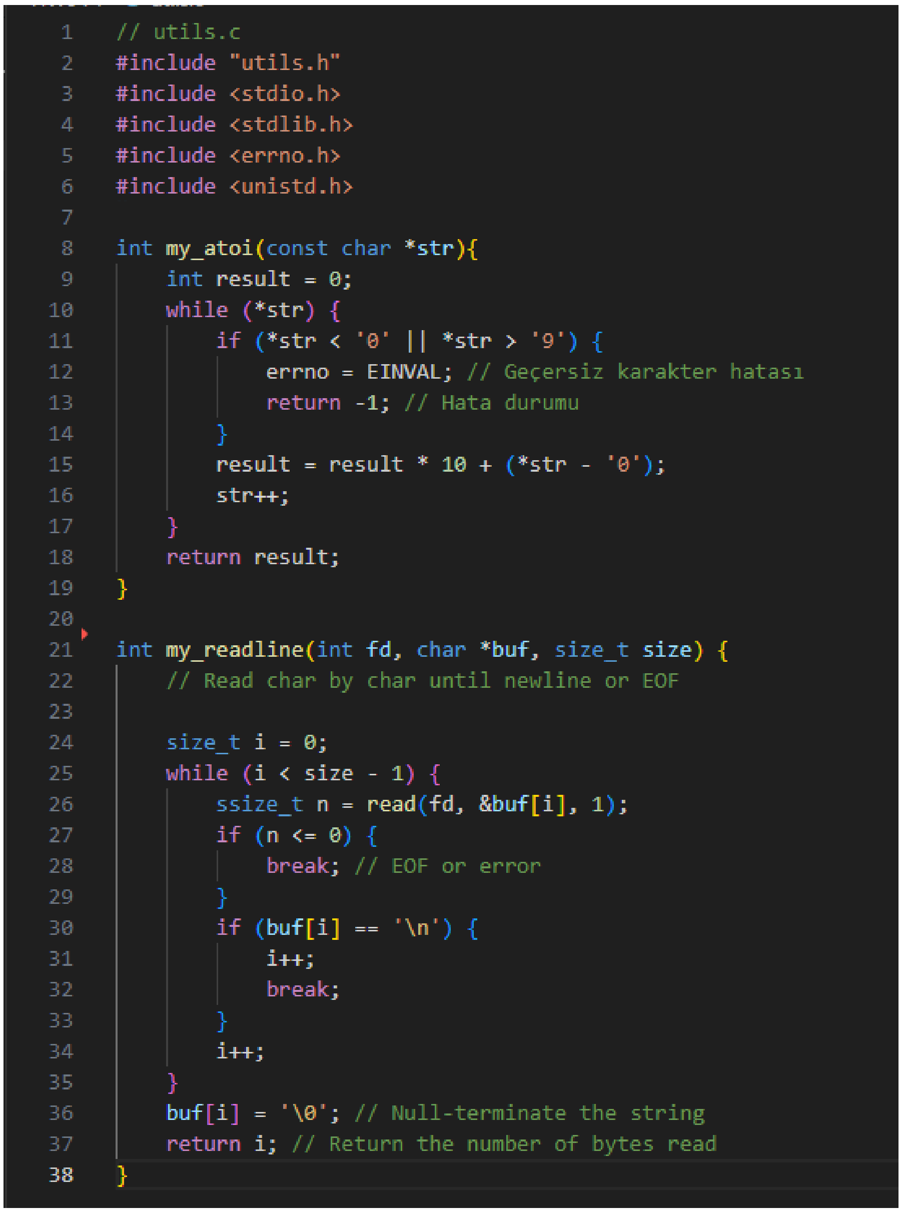
- **buffer.c / buffer.h**
  - Implements a bounded circular queue buffer with fixed capacity.
  - Provides buffer_init, buffer_add, buffer_get, and buffer_destroy functions.
  - Synchronization achieved via pthread_mutex_t and pthread_cond_t.
  - Handles full and empty buffer conditions without busy-waiting.



```c
// buffer.c
#include "buffer.h"
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

void buffer_init(SharedBuffer* buf, int capacity) {
    buf->lines = malloc(sizeof(char*) * capacity);
    buf->capacity = capacity;
    buf->count = 0;
    buf->front = 0;
    buf->rear = 0;
    buf->eof = 0;
    pthread_mutex_init(&buf->mutex, NULL);
    pthread_cond_init(&buf->not_empty, NULL);
    pthread_cond_init(&buf->not_full, NULL);
}

void buffer_destroy(SharedBuffer* buf) {
    for (int i = 0; i < buf->count; ++i) {
        int index = (buf->front + i) % buf->capacity;
        free(buf->lines[index]); // satırların belleğini boşalt
    }
    free(buf->lines);
    pthread_mutex_destroy(&buf->mutex);
    pthread_cond_destroy(&buf->not_empty);
    pthread_cond_destroy(&buf->not_full);
}

void buffer_add(SharedBuffer* buf, const char* line) {
    pthread_mutex_lock(&buf->mutex);
    while (buf->count == buf->capacity)
```

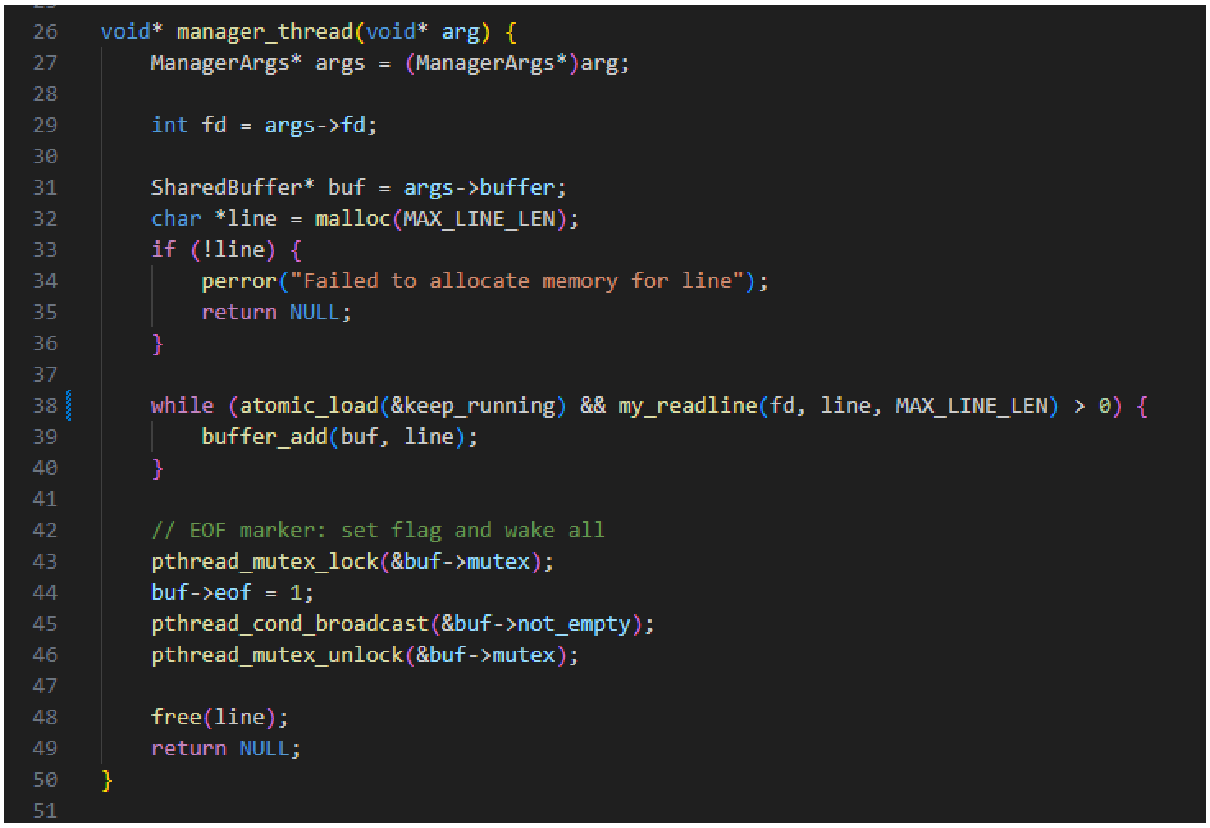
```c
void buffer_destroy(SharedBuffer* buf) {
}

void buffer_add(SharedBuffer* buf, const char* line) {
    pthread_mutex_lock(&buf->mutex);
    while (buf->count == buf->capacity)
        pthread_cond_wait(&buf->not_full, &buf->mutex);

    buf->lines[buf->rear] = strdup(line);
    buf->rear = (buf->rear + 1) % buf->capacity;
    buf->count++;
    pthread_cond_signal(&buf->not_empty);
    pthread_mutex_unlock(&buf->mutex);
}

char* buffer_get(SharedBuffer* buf) {
    pthread_mutex_lock(&buf->mutex);
    while (buf->count == 0 && !buf->eof)
        pthread_cond_wait(&buf->not_empty, &buf->mutex);

    if (buf->count == 0 && buf->eof) {
        pthread_mutex_unlock(&buf->mutex);
        return NULL;
    }

    char* line = buf->lines[buf->front];
    buf->front = (buf->front + 1) % buf->capacity;
    buf->count--;
    pthread_cond_signal(&buf->not_full);
    pthread_mutex_unlock(&buf->mutex);
    return line;
}
```

- **utils.c / utils.h**
  - Defines my_readline(fd, buf, size) function which reads a line from a file descriptor character by character, respecting newline or EOF boundaries.
  - Also defines my_atoi(const char *str) function which safely converts a string to integer, validating input and setting errno = EINVAL on invalid characters. Returns -1 on failure.



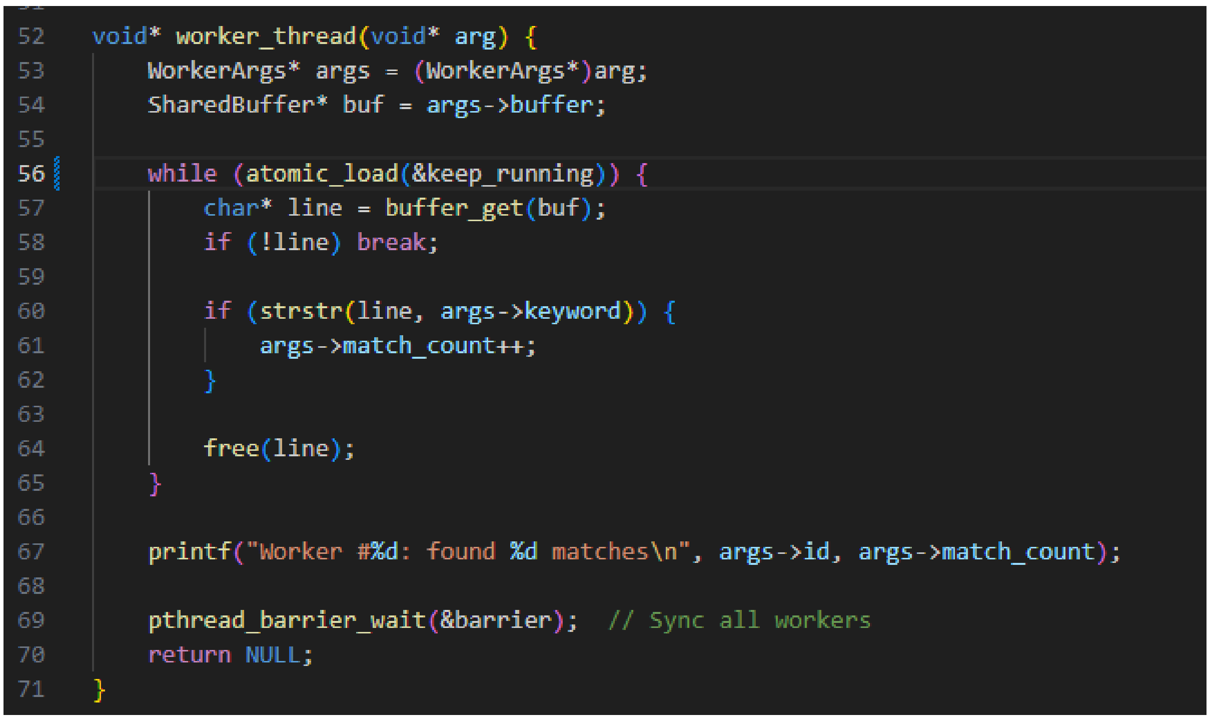
```c
// utils.c
#include "utils.h"
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <unistd.h>

int my_atoi(const char *str){
    int result = 0;
    while (*str) {
        if (*str < '0' || *str > '9') {
            errno = EINVAL; // Geçersiz karakter hatası
            return -1; // Hata durumu
        }
        result = result * 10 + (*str - '0');
        str++;
    }
    return result;
}

int my_readline(int fd, char *buf, size_t size) {
    // Read char by char until newline or EOF

    size_t i = 0;
    while (i < size - 1) {
        ssize_t n = read(fd, &buf[i], 1);
        if (n <= 0) {
            break; // EOF or error
        }
        if (buf[i] == '\n') {
            i++;
            break;
        }
        i++;
    }
    buf[i] = '\0'; // Null-terminate the string
    return i; // Return the number of bytes read
}
```

- **Manager Thread**
    - Continuously reads lines from the log file using my_readline().
    - Adds lines to the buffer using buffer_add().
    - Upon EOF or SIGINT, sets eof flag and wakes up all workers.



```c
void* manager_thread(void* arg) {
    ManagerArgs* args = (ManagerArgs*)arg;

    int fd = args->fd;

    SharedBuffer* buf = args->buffer;
    char *line = malloc(MAX_LINE_LEN);
    if (!line) {
        perror("Failed to allocate memory for line");
        return NULL;
    }

    while (atomic_load(&keep_running) && my_readline(fd, line, MAX_LINE_LEN) > 0) {
        buffer_add(buf, line);
    }

    // EOF marker: set flag and wake all
    pthread_mutex_lock(&buf->mutex);
    buf->eof = 1;
    pthread_cond_broadcast(&buf->not_empty);
    pthread_mutex_unlock(&buf->mutex);

    free(line);
    return NULL;
}
```

- ## Worker Thread
    - Continuously retrieves lines using buffer_get().
    - Searches for the keyword using strstr().
    - Tracks local match count.
    - Waits at the barrier after processing is done.
    - One worker prints the final total match count.

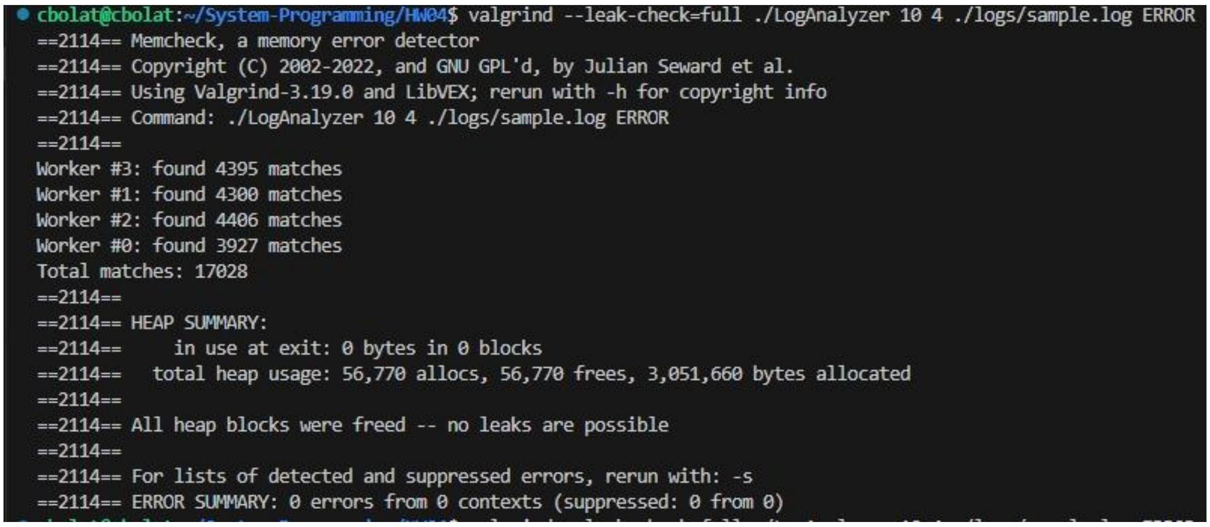


```c
void* worker_thread(void* arg) {
    WorkerArgs* args = (WorkerArgs*)arg;
    SharedBuffer* buf = args->buffer;

    while (atomic_load(&keep_running)) {
        char* line = buffer_get(buf);
        if (!line) break;

        if (strstr(line, args->keyword)) {
            args->match_count++;
        }

        free(line);
    }

    printf("Worker #%d: found %d matches\n", args->id, args->match_count);

    pthread_barrier_wait(&barrier);  // Sync all workers
    return NULL;
}
```

- ## Signal Handling
    - Uses SIGINT to interrupt the program gracefully.
    - Sets keep_running = 0 using pthread_mutex to avoid data races.
    - Ensures all memory is properly released.

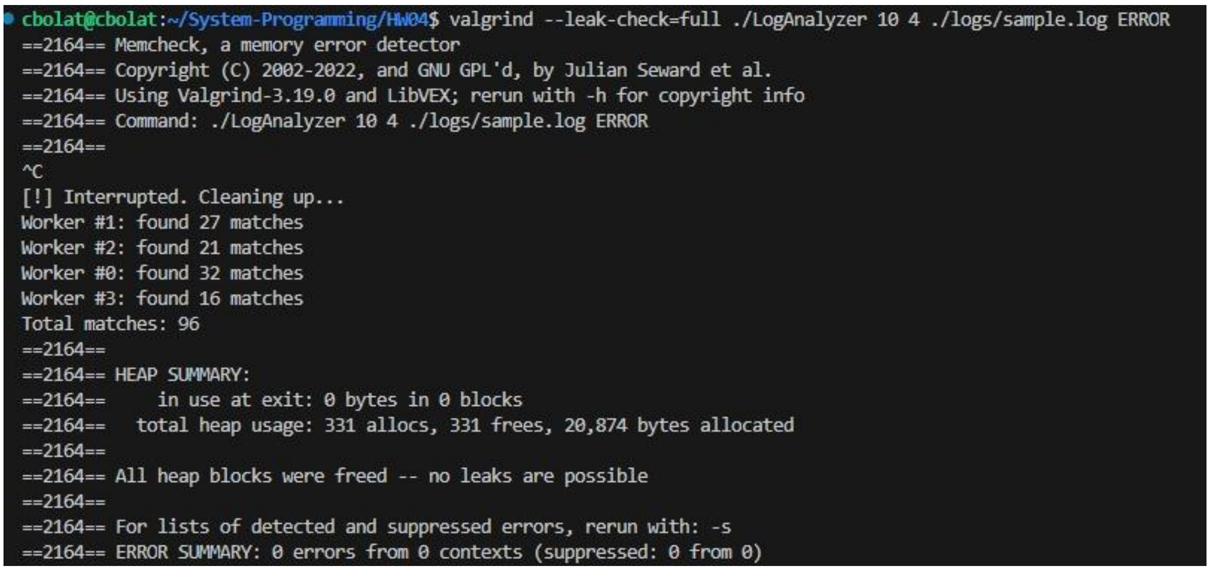
TESTING:

MemLeak check without interrupt signal.



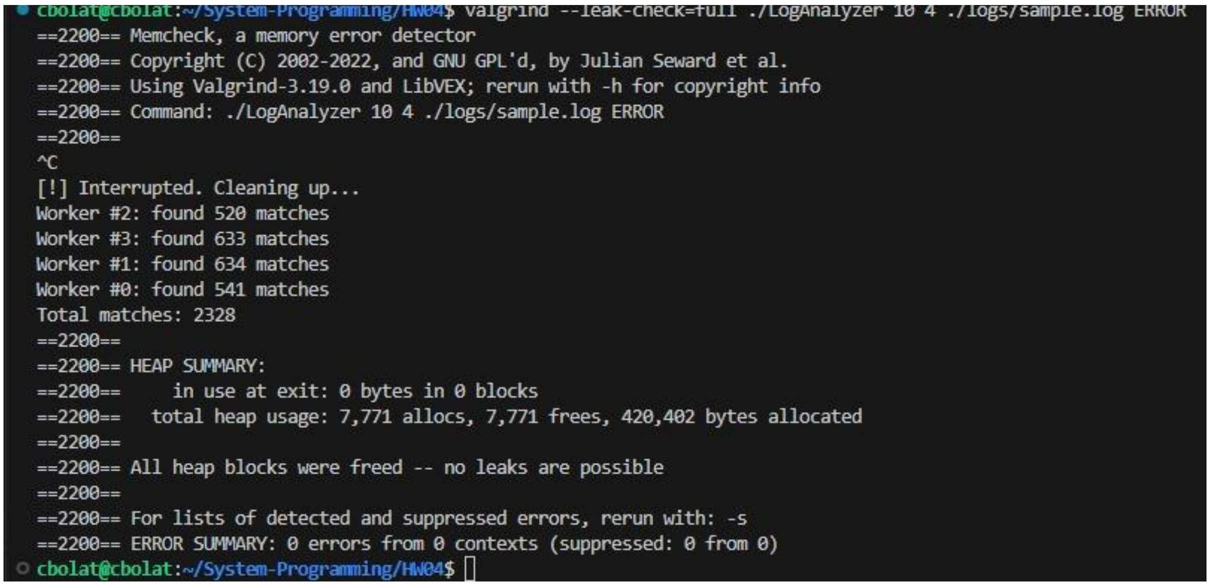
```
● cbolat@cbolat:~/System-Programming/HW04$ valgrind --leak-check=full ./LogAnalyzer 10 4 ./logs/sample.log ERROR
  ==2114== Memcheck, a memory error detector
  ==2114== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
  ==2114== Using Valgrind-3.19.0 and LibVEX; rerun with -h for copyright info
  ==2114== Command: ./LogAnalyzer 10 4 ./logs/sample.log ERROR
  ==2114==
  Worker #3: found 4395 matches
  Worker #1: found 4300 matches
  Worker #2: found 4406 matches
  Worker #0: found 3927 matches
  Total matches: 17028
  ==2114==
  ==2114== HEAP SUMMARY:
  ==2114==     in use at exit: 0 bytes in 0 blocks
  ==2114==   total heap usage: 56,770 allocs, 56,770 frees, 3,051,660 bytes allocated
  ==2114==
  ==2114== All heap blocks were freed -- no leaks are possible
  ==2114==
  ==2114== For lists of detected and suppressed errors, rerun with: -s
  ==2114== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

MemLeak check with interrupt signal



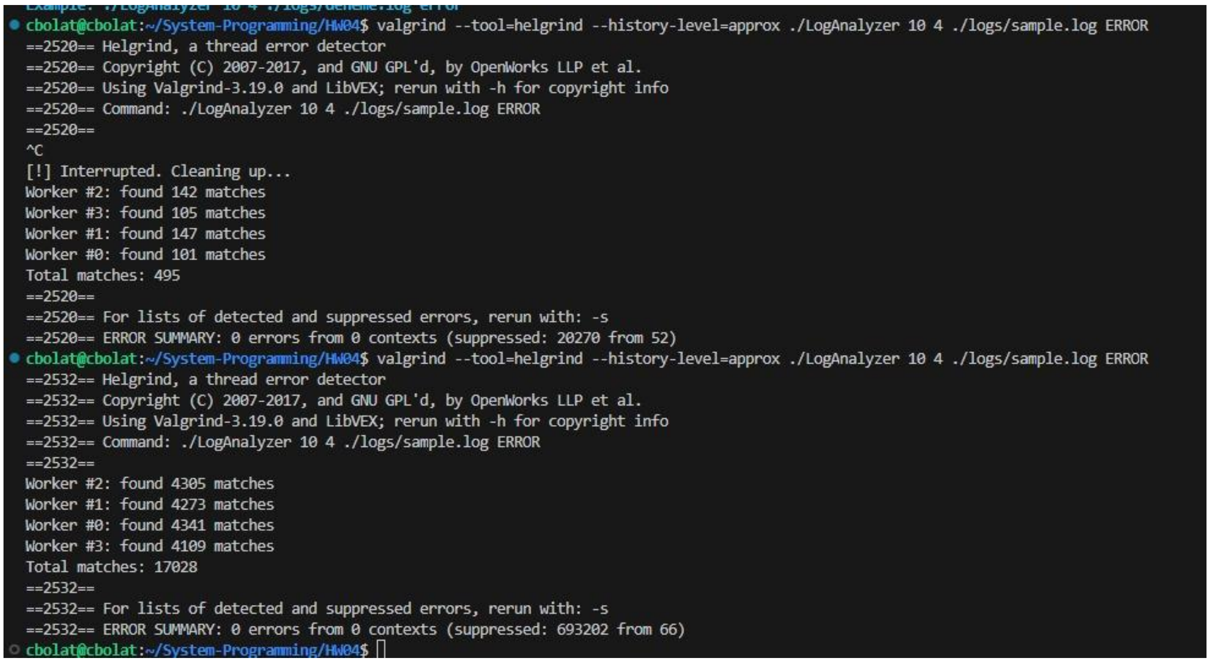
```
● cbolat@cbolat:~/System-Programming/HW04$ valgrind --leak-check=full ./LogAnalyzer 10 4 ./logs/sample.log ERROR
  ==2164== Memcheck, a memory error detector
  ==2164== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
  ==2164== Using Valgrind-3.19.0 and LibVEX; rerun with -h for copyright info
  ==2164== Command: ./LogAnalyzer 10 4 ./logs/sample.log ERROR
  ==2164==
  ^C
  [!] Interrupted. Cleaning up...
  Worker #1: found 27 matches
  Worker #2: found 21 matches
  Worker #0: found 32 matches
  Worker #3: found 16 matches
  Total matches: 96
  ==2164==
  ==2164== HEAP SUMMARY:
  ==2164==     in use at exit: 0 bytes in 0 blocks
  ==2164==   total heap usage: 331 allocs, 331 frees, 20,874 bytes allocated
  ==2164==
  ==2164== All heap blocks were freed -- no leaks are possible
  ==2164==
  ==2164== For lists of detected and suppressed errors, rerun with: -s
  ==2164== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Data race check with interrupt signal.



```
cbolat@cbolat:~/System-Programming/HW04$ valgrind --leak-check=full ./LogAnalyzer 10 4 ./logs/sample.log ERROR
==2200== Memcheck, a memory error detector
==2200== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==2200== Using Valgrind-3.19.0 and LibVEX; rerun with -h for copyright info
==2200== Command: ./LogAnalyzer 10 4 ./logs/sample.log ERROR
==2200==
^C
[!] Interrupted. Cleaning up...
Worker #2: found 520 matches
Worker #3: found 633 matches
Worker #1: found 634 matches
Worker #0: found 541 matches
Total matches: 2328
==2200==
==2200== HEAP SUMMARY:
==2200==     in use at exit: 0 bytes in 0 blocks
==2200==   total heap usage: 7,771 allocs, 7,771 frees, 420,402 bytes allocated
==2200==
==2200== All heap blocks were freed -- no leaks are possible
==2200==
==2200== For lists of detected and suppressed errors, rerun with: -s
==2200== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
cbolat@cbolat:~/System-Programming/HW04$
```

Data race check with/without interrupt signal.



```
cbolat@cbolat:~/System-Programming/HW04$ valgrind --tool=helgrind --history-level=approx ./LogAnalyzer 10 4 ./logs/sample.log ERROR
==2520== Helgrind, a thread error detector
==2520== Copyright (C) 2007-2017, and GNU GPL'd, by OpenWorks LLP et al.
==2520== Using Valgrind-3.19.0 and LibVEX; rerun with -h for copyright info
==2520== Command: ./LogAnalyzer 10 4 ./logs/sample.log ERROR
==2520==
^C
[!] Interrupted. Cleaning up...
Worker #2: found 142 matches
Worker #3: found 105 matches
Worker #1: found 147 matches
Worker #0: found 101 matches
Total matches: 495
==2520==
==2520== For lists of detected and suppressed errors, rerun with: -s
==2520== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 20270 from 52)
cbolat@cbolat:~/System-Programming/HW04$ valgrind --tool=helgrind --history-level=approx ./LogAnalyzer 10 4 ./logs/sample.log ERROR
==2532== Helgrind, a thread error detector
==2532== Copyright (C) 2007-2017, and GNU GPL'd, by OpenWorks LLP et al.
==2532== Using Valgrind-3.19.0 and LibVEX; rerun with -h for copyright info
==2532== Command: ./LogAnalyzer 10 4 ./logs/sample.log ERROR
==2532==
Worker #2: found 4305 matches
Worker #1: found 4273 matches
Worker #0: found 4341 matches
Worker #3: found 4109 matches
Total matches: 17028
==2532==
==2532== For lists of detected and suppressed errors, rerun with: -s
==2532== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 693202 from 66)
cbolat@cbolat:~/System-Programming/HW04$
```

Conclusion:

This assignment required deep understanding of multithreading, synchronization, and memory safety in C. The key challenges included safely handling shared memory between threads, avoiding deadlocks, and preventing data races—especially when SIGINT was received during processing. These were resolved by using condition variables and barriers correctly, and by protecting shared flags like keep_running with a mutex. Additional effort was made to ensure graceful shutdown and memory deallocation in all edge cases. The project successfully demonstrated efficient log analysis using multithreaded design principles.