CSE344 - Homework#4

Muhammet Talha Memişoğlu

210104004009

Introduction

This project implements a multithreaded log file analysis system using the POSIX threading library in C. The goal is to search for a specific keyword within a log file using concurrent worker threads that consume data from a bounded buffer. The manager (main) thread reads the file line by line and populates a shared buffer, while worker threads consume and process these lines concurrently. Key aspects of the project include thread synchronization, signal handling, and inter-thread communication through condition variables and mutexes. This project was tested on Debain 11 and 12.

Code Explanation

Buffer Module

The buffer system, defined in buffer.c and buffer.h, is responsible for storing lines read from the log file and facilitating safe communication between the producer (manager thread) and consumers (worker threads) with mutexes and condition variables.

- **Initialization (buffer_init())**
  Allocates memory and initializes synchronization primitives.
- **Destruction (buffer_destroy())**
  Frees allocated memory and destroys primitives.
- **Put (buffer_put)**
  Waits if the buffer is full, then inserts a line. This operation is protected with mutex and condition variable.

```
pthread_mutex_lock(&buffer->mutex);

// Wait until there is space in the buffer
while (buffer->count == buffer->size) {
    pthread_cond_wait(&buffer->not_full, &buffer->mutex);
}
```

- **Get (buffer_get)**
  Waits if the buffer is empty, then retrieves a line. This operation is protected with mutex and condition variable.

```c
pthread_mutex_lock(&buffer->mutex);

// Wait until there is data in the buffer
while (buffer->count == 0) {
    pthread_cond_wait(&buffer->not_empty, &buffer->mutex);
}
```

Main Module

The main logic is defined in main.c and consists of these components:

- **Signal Handling**

  A signal handler for SIGINT is registered using sigaction. When Ctrl+C is pressed, a global flag (terminate) is set, allowing threads to exit gracefully. It prevents memory leaks or deadlocks.

```c
// Signal handler for SIGINT
void handle_sigint(int sig) {
    terminate = 1;
}
```

- **Manager Thread (Producer)**
  The main thread reads the input file using getline, creates a deep copy of each line, and inserts it into the buffer using buffer_put. After reading all lines, it inserts a NULL as a sentinel value to indicate EOF to workers.

```c
// Read file line by line
while ((read = getline(&line, &len, file)) != -1 && !terminate) {
    // Create a copy of the line
    char *line_copy = strdup(line);
    if (line_copy == NULL) {
        perror("Failed to allocate memory for line");
        continue;
    }

    // Put the line into the buffer
    buffer_put(shared_buffer, line_copy);
}
```

- **Worker Threads (Consumers)**
  Each worker thread performs the following:
  - Continuously retrieves lines from the buffer using buffer_get().
  - Searches for the given keyword using strstr().

- Increments a local counter if a match is found.
- Uses a barrier to synchronize the reporting phase.
- Adds the local count to a shared total count (total_matches) using a mutex.
- Each worker exits when it receives a NULL value from the buffer.
- Barrier usage ensures that all workers finish processing before a summary is printed.

```c
// Consume from buffer
line = buffer_get(shared_buffer);

// Check for EOF marker
if (line == NULL) {
    // Put EOF marker back for other workers and exit
    buffer_put(shared_buffer, NULL);
    break;
}

// Search for keyword
if (strstr(line, search_term) != NULL) {
    matches_found++;
    printf("Worker %d found match: %s", worker_id, line);
}
```

- **Synchronization Techniques**
  - **Mutexes:** Used to protect the shared buffer and total match count.
  - **Condition Variables:** Enable the producer and consumers to wait and signal based on buffer state.
  - **Barriers:** Ensure that all threads reach a synchronization point before concluding the analysis.

- **Testing and Execution**
  The program is executed using:

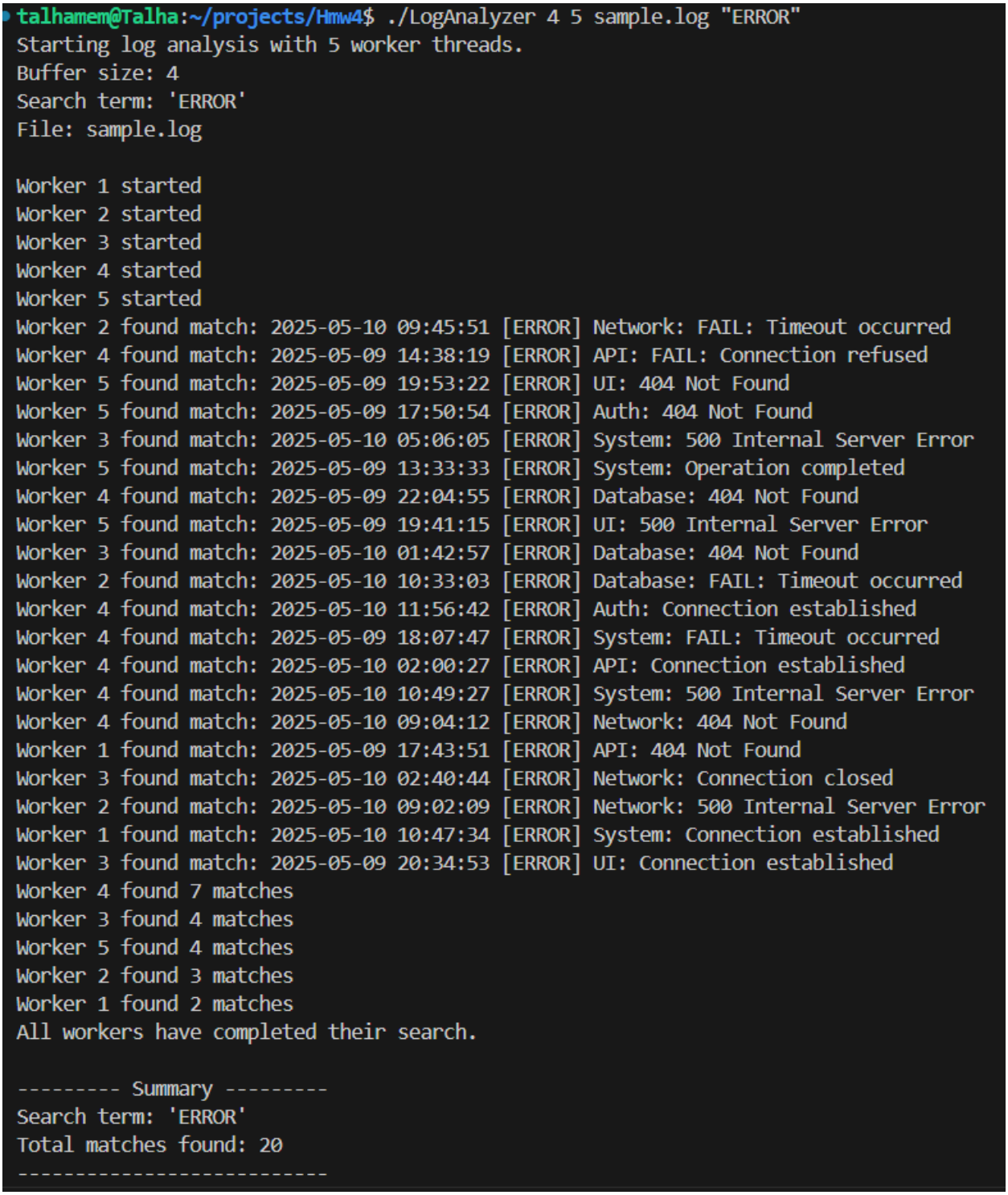
  ./log_analyzer <buffer_size> <num_workers> <log_file> <search_term>

  For example:

  ./log_analyzer 10 4 sample.log "ERROR"

  Interrupting the program with Ctrl+C triggers a graceful shutdown, preserving partial results and avoiding memory leaks or deadlocks.

Screenshots



```
talhamem@Talha:~/projects/Hmw4$ ./LogAnalyzer 20 5 sample.log "ERROR"
Starting log analysis with 5 worker threads.
Buffer size: 20
Search term: 'ERROR'
File: sample.log

Worker 1 started
Worker 2 started
Worker 3 started
Worker 4 started
Worker 5 started
Worker 2 found match: 2025-05-10 09:45:51 [ERROR] Network: FAIL: Timeout occurred
Worker 3 found match: 2025-05-09 14:38:19 [ERROR] API: FAIL: Connection refused
Worker 4 found match: 2025-05-09 19:53:22 [ERROR] UI: 404 Not Found
Worker 4 found match: 2025-05-09 13:33:33 [ERROR] System: Operation completed
Worker 2 found match: 2025-05-10 05:06:05 [ERROR] System: 500 Internal Server Error
Worker 3 found match: 2025-05-09 19:41:15 [ERROR] UI: 500 Internal Server Error
Worker 5 found match: 2025-05-09 22:04:55 [ERROR] Database: 404 Not Found
Worker 1 found match: 2025-05-09 17:50:54 [ERROR] Auth: 404 Not Found
Worker 1 found match: 2025-05-10 10:33:03 [ERROR] Database: FAIL: Timeout occurred
Worker 2 found match: 2025-05-10 01:42:57 [ERROR] Database: 404 Not Found
Worker 1 found match: 2025-05-09 17:43:51 [ERROR] API: 404 Not Found
Worker 1 found match: 2025-05-10 02:00:27 [ERROR] API: Connection established
Worker 1 found match: 2025-05-10 10:49:27 [ERROR] System: 500 Internal Server Error
Worker 1 found match: 2025-05-10 02:40:44 [ERROR] Network: Connection closed
Worker 1 found match: 2025-05-10 09:02:09 [ERROR] Network: 500 Internal Server Error
Worker 1 found match: 2025-05-10 09:04:12 [ERROR] Network: 404 Not Found
Worker 1 found match: 2025-05-10 10:47:34 [ERROR] System: Connection established
Worker 3 found match: 2025-05-09 18:07:47 [ERROR] System: FAIL: Timeout occurred
Worker 1 found match: 2025-05-09 20:34:53 [ERROR] UI: Connection established
Worker 4 found match: 2025-05-10 11:56:42 [ERROR] Auth: Connection established
Worker 5 found 1 matches
Worker 4 found 3 matches
Worker 1 found 10 matches
Worker 3 found 3 matches
Worker 2 found 3 matches
All workers have completed their search.

--------- Summary ---------
Search term: 'ERROR'
Total matches found: 20
---------------------------
```

Testing for memory leak

```
talhamem@Talha:~/projects/Hmw4$ valgrind ./LogAnalyzer 20 5 sample.log "ERROR"
==27252== Memcheck, a memory error detector
==27252== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==27252== Using Valgrind-3.19.0 and LibVEX; rerun with -h for copyright info
==27252== Command: ./LogAnalyzer 20 5 sample.log ERROR
==27252==
Starting log analysis with 5 worker threads.
Buffer size: 20
Search term: 'ERROR'
File: sample.log

Worker 2 started
Worker 1 started
Worker 3 started
Worker 3 found match: 2025-05-09 14:38:19 [ERROR] API: FAIL: Connection refused
Worker 3 found match: 2025-05-09 19:53:22 [ERROR] UI: 404 Not Found
Worker 4 started
Worker 5 started
Worker 5 found match: 2025-05-09 13:33:33 [ERROR] System: Operation completed
Worker 5 found match: 2025-05-09 19:41:15 [ERROR] UI: 500 Internal Server Error
```

.

.

.

```
Worker 2 found 1 matches
All workers have completed their search.

--------- Summary ---------
Search term: 'ERROR'
Total matches found: 20
---------------------------
==27252==
==27252== HEAP SUMMARY:
==27252==     in use at exit: 0 bytes in 0 blocks
==27252==   total heap usage: 67 allocs, 67 frees, 10,345 bytes allocated
==27252==
==27252== All heap blocks were freed -- no leaks are possible
==27252==
==27252== For lists of detected and suppressed errors, rerun with: -s
==27252== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Conclusion

In this project, I created a program that uses multiple threads to search for a word in a log file. The main thread reads lines from the file and puts them into a shared buffer. Worker threads take lines from the buffer and look for the keyword. If they find a match, they count it. All threads work together safely using mutexes and condition variables.

A signal handler was added so the program can stop safely if the user presses Ctrl+C. Also, barriers are used to make sure all worker threads finish before the result is shown.

This project helped me understand how to use threads, share data safely, and make sure all parts of a multithreaded program run correctly and finish properly. It also showed how to deal with user interruptions.