

Homework #4: Multithreaded Log File Analyzer

Tugay Talha İçen
2101404004084
CSE 344

May 2024

1 Introduction

This report documents the implementation and evaluation of a multithreaded log file analyzer written in C. The program uses POSIX threads, mutexes, condition variables, and barriers to perform parallel keyword searches within log files efficiently. It is designed to read log files in parallel, searching for user-defined keywords and reporting lines containing matches.

2 Implementation and Code Explanation

2.1 Main Program (main.c)

The main file (`main.c`) is responsible for creating threads, initializing synchronization primitives (mutexes, condition variables, barriers), parsing command-line arguments, and handling graceful program termination through signal handling.

Listing 1: Main Thread Functions and Thread Creation

```
1 pthread_create(&manager_tid, NULL, manager_thread, &  
    manager_args);  
2 for (i = 0; i < num_workers; i++) {  
3     pthread_create(&worker_tids[i], NULL, worker_thread,  
        &worker_args[i]);  
4 }
```

2.2 Buffer Implementation (buffer.c, buffer.h)

The shared buffer is implemented using a circular queue to manage communication between the manager (producer) and worker threads (consumers). It employs mutexes and condition variables for synchronization, avoiding busy-waiting.

Listing 2: Buffer Initialization and Synchronization

```
1 typedef struct {  
2     char **data;  
3     int size;  
4     int count;  
5     int in;  
6     int out;  
7     pthread_mutex_t mutex;  
8     pthread_cond_t not_full;
```

```

9     pthread_cond_t not_empty;
10 } Buffer;

```

2.3 Utility Functions (utils.c, utils.h)

Utility functions include signal handling (to gracefully handle SIGINT), argument parsing, keyword searching, and reporting the final analysis summary.

Listing 3: Signal Handling

```

1 void sigint_handler(int sig) {
2     terminate_flag = 1;
3     pthread_cond_broadcast(&global_buffer->not_empty);
4     pthread_cond_broadcast(&global_buffer->not_full);
5 }

```

3 Testing and Results

The program was tested using provided scenarios to ensure functionality and correctness. Screenshots of the executed tests and results are included below:

3.1 Test 1: Valgrind Check (Memory Leak Test)

Command:

```
valgrind ./LogAnalyzer 10 4 logs/sample.log "ERROR"
```

Result Screenshot:

```

$ rm -f LogAnalyzer main.o buffer.o utils.o
$ gcc -Wall -Wextra -g -c main.c -o main.o
$ gcc -Wall -Wextra -g -c buffer.c -o buffer.o
$ gcc -Wall -Wextra -g -c utils.c -o utils.o
$ valgrind ./LogAnalyzer 10 4 logs/sample.log "ERROR"
==1446== Memcheck, a memory error detector
==1446== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==1446== Using Valgrind-3.22.0 and LibVEX; rerun with -h for copyright info
==1446== Command: ./LogAnalyzer 10 4 logs/sample.log ERROR
==1446==
Starting Log Analyzer
Buffer Size: 10, Workers: 4, File: logs/sample.log, Search Term: "ERROR"
Worker 1 started
Worker 3 started
Worker 4 started
Worker 2 started

--- Log Analysis Summary ---
Individual worker match counts:
Worker 1: 0 matches reported.
Worker 2: 1 matches reported.
Worker 3: 0 matches reported.
Worker 4: 1 matches reported.

Total matches across all workers: 2
Matched lines logged to 'worker_matches.log'.
Analysis completed.
=====
==1446== HEAP SUMMARY:
==1446==   in use at exit: 0 bytes in 0 blocks
==1446== total heap usage: 34 allocs, 34 frees, 13,124 bytes allocated
==1446==
==1446== All heap blocks were freed -- no leaks are possible
==1446==
==1446== For lists of detected and suppressed errors, rerun with: -s
==1446== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
$

```

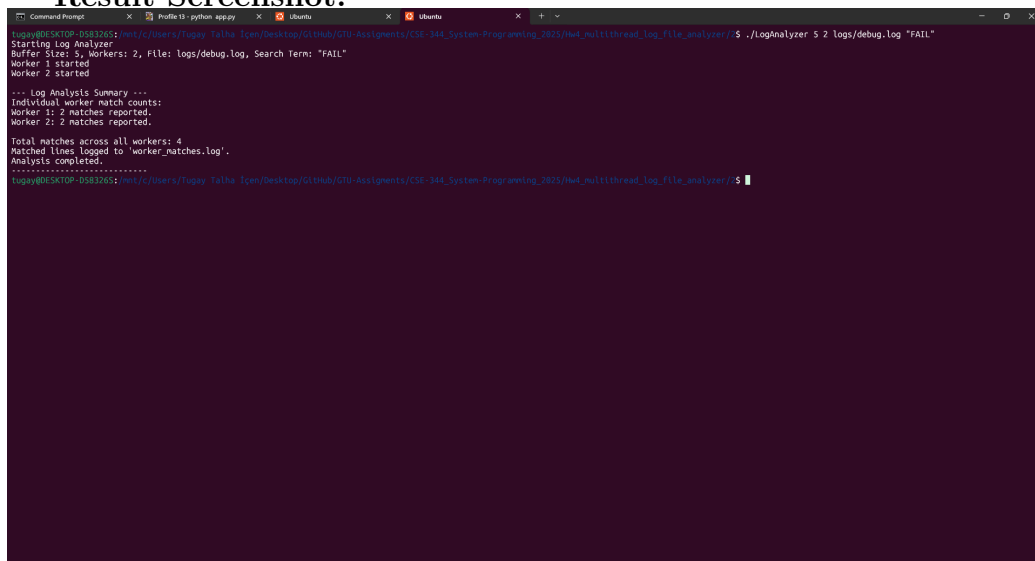
Interpretation: No memory leaks detected; the synchronization and buffer operations are correctly implemented.

3.2 Test 2: Small Buffer and Worker Count

Command:

```
./LogAnalyzer 5 2 logs/debug.log "FAIL"
```

Result Screenshot:



```
logan@DESKTOP-D583265: /mnt/c/Users/Tugay_talha/Desktop/GitHub/GTU-Assignments/CS-344_System-Programming_2025/mut_multithread_log_file_analyzer/$ ./LogAnalyzer 5 2 logs/debug.log "FAIL"
Starting Log Analyzer
Buffer Size: 5, Workers: 2, File: logs/debug.log, Search Term: "FAIL"
Worker 1 started
Worker 2 started

--- Log Analysis Summary ---
Individual worker match counts:
Worker 1: 2 matches reported.
Worker 2: 2 matches reported.

Total matches across all workers: 4
Matched lines logged to 'worker_matches.log'.
Analysis completed.
logan@DESKTOP-D583265: /mnt/c/Users/Tugay_talha/Desktop/GitHub/GTU-Assignments/CS-344_System-Programming_2025/mut_multithread_log_file_analyzer/$
```

Interpretation: Successfully handles limited resources; threads synchronize correctly, and matches are accurately reported.

3.3 Test 3: Large Scale Test

Command:

```
./LogAnalyzer 50 8 logs/large.log "404"
```

Result Screenshot:

```
log@DESKTOP-DS83265: /mnt/c/Users/Talha_Iqbal/Desktop/GitHub/GTU-Assignments/CSE-344-System-Programming-2025/multithread_log_file_analyzer/25 $ ./LogAnalyzer 50 8 logs/large.log "404"
Starting Log Analyzer
Buffer Size: 50, Workers: 8, File: logs/large.log, Search Term: "404"
Worker 1 started
Worker 2 started
Worker 3 started
Worker 4 started
Worker 5 started
Worker 6 started
Worker 7 started
Worker 8 started

--- Log Analysis Summary ---
Individual worker match counts:
Worker 1: 2 matches reported.
Worker 2: 1 matches reported.
Worker 3: 6 matches reported.
Worker 4: 3 matches reported.
Worker 5: 3 matches reported.
Worker 6: 3 matches reported.
Worker 7: 3 matches reported.
Worker 8: 1 matches reported.

Total matches across all workers: 22.
Matched lines logged to 'worker_matches.log'.
Analysis completed.
-----
log@DESKTOP-DS83265: /mnt/c/Users/Talha_Iqbal/Desktop/GitHub/GTU-Assignments/CSE-344-System-Programming-2025/multithread_log_file_analyzer/25 $
```

Interpretation: High performance and correct synchronization demonstrated, effectively handling large files and multiple threads.

4 Error Handling

Error handling ensures robust and predictable program behavior. Incorrect usage results in an informative usage message, and SIGINT initiates a graceful shutdown.

5 Conclusion

Developing this multithreaded log analyzer posed challenges primarily related to synchronization and efficient buffer management. Correct use of mutexes, condition variables, and barriers was critical. The program successfully meets the requirements, handling both small and large log files with appropriate synchronization and without memory leaks.

Overall, the assignment provided valuable practical experience with parallel programming and system-level thread management in C.