007

Threads II

Operating Systems

Lab007 Threads II

(Multi-Threaded Web Log Analyzer)

Objective

The primary goal of this lab is to teach students how to use **multi-threading** to process **large text-based log files** efficiently. By parallelizing the computation, students will **reduce execution time** and **understand the real-world benefits of multi-threading**.

Scenario: Multi-Threaded Web Log Analyzer

A web server generates large log files containing millions of records. Each record contains information about an HTTP request, including the IP address, timestamp, request type, status code, and URL.
Your task is to write a multi-threaded log analyzer that:

1. Counts the number of requests made by each unique IP address.
2. Identifies the most frequently visited URL.
3. Finds the number of HTTP errors (status codes 400–599).

To speed up processing, the program will:

- Use multiple threads, each handling a portion of the log file.
- Merge results from all threads (**Synchronization** may be needed).

Expected Output
For a log file (access.log) like:
192.168.1.1 - - [10/Feb/2024:10:20:30] "GET /index.html" 200
192.168.1.2 - - [10/Feb/2024:10:20:31] "POST /login" 403
192.168.1.1 - - [10/Feb/2024:10:20:32] "GET /about.html" 200
192.168.1.3 - - [10/Feb/2024:10:20:33] "GET /index.html" 200
192.168.1.2 - - [10/Feb/2024:10:20:34] "GET /index.html" 404
The program should output:
Total Unique IPs: 3
Most Visited URL: /index.html (3 times)
HTTP Errors: 2

Project Structure:

```
├── main_linux.c   (Main program for Linux using pthreads)
├── main_windows.c (Main program for Windows using WinAPI)
├── log_processor.h (Header file with function prototypes)
├── log_processor.c (Functions for log processing)
├── Makefile       (For compiling the Linux version)
├── run.sh         (Shell script to run the Linux version)
├── run.bat        (Batch script to run the Windows version)
├── access.log     (Sample web log file)
```

Assignment Tasks

1. **Implement Threaded Log Processing:**
   - Read log file chunks in parallel.
   - Count unique IPs, URLs, and errors.

2. **Merge Thread Results:**
   - Combine data from each thread.

3. **Optimize Performance:**
   - Use efficient data structures (hash tables).
   - Process large files in parallel.

4. **Print the Results**
   - Unique IP count
   - Most visited URL
   - Total HTTP errors

5. **Test with Different Log Files:**
   - Provide students with small & large logs.
   - Compare execution time for single-threaded vs multi-threaded.