Universidade Presbiteriana Mackenzie

Faculdade de Computação e Informática - FCI

Projeto Prático 1 (LAB1)
Analisador de Logs de Alta Performance com Pthreads

Computação Paralela

Prof. Dr. Jean M. Laine

1. Introdução: A Importância da Análise de Logs

Em qualquer sistema de software moderno, desde servidores web até aplicações distribuídas complexas, os arquivos de log são a principal fonte de verdade para entender o que está acontecendo. Eles registram cada requisição, cada erro e cada evento crítico. Com o volume de dados gerados hoje, analisar esses logs de forma eficiente deixou de ser uma tarefa trivial e se tornou um problema de *Big Data*. Um administrador de sistemas que precisa identificar a origem de um ataque em tempo real ou um engenheiro de software que busca otimizar a performance de uma aplicação não podem esperar horas por um relatório. A velocidade na extração de informações (estatísticas, padrões de erro, anomalias) é crucial. É neste cenário que a computação paralela se torna indispensável. Este projeto irá simular exatamente este desafio do mundo real: você desenvolverá um analisador de logs de servidor web capaz de processar milhões de registros em segundos, utilizando o poder do paralelismo com Pthreads para acelerar drasticamente a tarefa.

2. Objetivos do Projeto

Ao final deste projeto, você será capaz de:

- Desenvolver uma aplicação em C do início ao fim para resolver um problema de processamento de dados.
- Implementar uma solução paralela utilizando a biblioteca Pthreads.
- Identificar seções críticas em código concorrente e protegê-las utilizando mutex para evitar condições de corrida.
- Realizar uma análise de desempenho completa, medindo o tempo de execução e calculando métricas como *Speedup* e Eficiência.
- Argumentar sobre os resultados obtidos, analisando o impacto e o *overhead* da sincronização no desempenho da aplicação.

3. Descrição do Problema

Você deve criar um programa em C que leia um arquivo de log de servidor web no formato *Common Log Format* e extraia duas estatísticas principais:

1. **Contagem de erros 404:** O número total de requisições que resultaram em um código de status "Not Found"(404).
2. **Total de bytes transferidos:** A soma total de todos os bytes transferidos em requisições bem-sucedidas (código 200).

O programa deve ser implementado em duas versões: uma **sequencial** e uma **paralela** (com Pthreads), permitindo uma comparação direta de desempenho. **Formato de uma linha do log (exemplo):**

```
127.0.0.1 - - [15/Sep/2025:15:30:00 -0300] "GET /index.html HTTP/1.1" 200 1500
127.0.0.1 - - [15/Sep/2025:15:30:01 -0300] "GET /nonexistent.jpg HTTP/1.1" 404 250
```

As informações relevantes são o **código de status** (ex: 200, 404) e o **número de bytes** transferidos (ex: 1500, 250).

4. Etapas do Projeto

Etapa 1: Obtenção dos Dados e Preparação do Ambiente

1. **Arquivo de Log:** Baixe o arquivo `generate_log.py` no Moodle e use ele para gerar o arquivo de log de exemplo `access_log_large.txt`. Este script irá gerar um arquivo com milhões de linhas. Este arquivo de log será grande o suficiente para que a diferença de desempenho entre as versões (sequencial e paralela) seja clara.
2. **Ambiente:** O projeto deve ser desenvolvido em um ambiente Linux/Unix, compilado com o GCC e utilizando a flag `-pthread`. Crie um `Makefile` para automatizar a compilação das duas versões do seu programa (sequencial e paralela).

Etapa 2: Implementação da Versão Sequencial

Crie um programa `log_analyzer_seq.c`. Este programa deve:

1. Abrir o arquivo de log para leitura.
2. Ler o arquivo linha por linha.
3. Para cada linha, extrair (fazer o *parse*) o código de status e o número de bytes.
4. Atualizar os contadores de erros 404 e o somatório de bytes.
5. Ao final, imprimir na tela o tempo total de execução e as estatísticas encontradas.

Etapa 3: Implementação da Versão Paralela com Pthreads

Crie um programa `log_analyzer_par.c`. Este programa deve:

1. Receber o número de threads a serem utilizadas como argumento de linha de comando.
2. Ler todas as linhas do arquivo para um array na memória.

3. Dividir o trabalho: determinar quais linhas cada thread será responsável por processar.
4. Criar uma estrutura de dados global para armazenar as estatísticas (e.g., `struct Stats { long long errors404; long long total_bytes; };`).
5. Criar um `pthread_mutex_t` global para proteger o acesso a essa estrutura.
6. Disparar as `N` threads. Cada thread deve processar sua fatia de linhas e, ao encontrar uma estatística, **travar o mutex**, atualizar a estrutura global e **destravar o mutex**.
7. A thread principal deve esperar (`pthread_join`) todas as threads terminarem.
8. Ao final, imprimir o tempo de execução e as estatísticas finais.

Etapa 4: Análise de Desempenho e Escalabilidade

1. Meça o tempo de execução da versão sequencial. Este será seu $T_{sequencial}$.
2. **Realize uma análise de Escalabilidade Forte (*Strong Scaling*):** Meça o tempo de execução da versão paralela ($T_{paralelo}$) para diferentes números de threads (obrigatororiamente para 1, 2, 4 e 8 threads). Em cada teste, lembre-se de usar sempre o mesmo arquivo de log (`access_log_large.txt`).
3. Organize os resultados em uma tabela contendo: Nº de Threads, Tempo de Execução (s), Speedup (S) e Eficiência (E).
4. Gere um gráfico de Speedup vs. Nº de Threads.

5. Dicas e Recursos Técnicos

Lendo um arquivo linha por linha em C: A função ‘getline‘ é a forma mais segura e recomendada para ler linhas de um arquivo.

```c
FILE *fp = fopen("access_log.txt", "r");
if (fp == NULL) {
  perror("Erro ao abrir o arquivo");
  exit(EXIT_FAILURE);
}

char *line = NULL;
size_t len = 0;
ssize_t read;

while ((read = getline(&line, &len, fp)) != -1) {
  // Processar a 'line' aqui...
}

fclose(fp);
if (line) {
  free(line);
}

```

Extraindo informações da linha (Parsing): Você pode usar ‘sscanf‘ para extrair os dados diretamente, mas ele pode ser frágil. Uma alternativa mais robusta é usar ‘strstr‘ para encontrar a parte da string que contém o código de status e os bytes, e então ‘sscanf‘ ou ‘strtol‘ para converter.

```c
// Exemplo simples para extrair código e bytes
// Pode ser necessário ajustar com base no formato exato
char *quote_ptr = strstr(line, "\" "); // Encontra o " depois da requisição
if (quote_ptr) {
  int status_code;
  long long bytes_sent;
  if (sscanf(quote_ptr + 2, "%d %lld", &status_code, &bytes_sent) == 2) {
    // Agora você tem os valores em status_code e bytes_sent
  }
}

```

Medindo o tempo com `clock_gettime`:

```c
#include <time.h>

struct timespec start, end;
clock_gettime(CLOCK_MONOTONIC, &start);

// --- SEU CÓDIGO AQUI ---

clock_gettime(CLOCK_MONOTONIC, &end);

double time_spent = (end.tv_sec - start.tv_sec) +
(end.tv_nsec - start.tv_nsec) / 1e9;
printf("Tempo de execução: %.4f segundos\n", time_spent);

```

6. Formato da Entrega e Avaliação

A entrega do projeto deve ser um arquivo '.zip' contendo:

1. O código-fonte dos programas `log_analyzer_seq.c` e `log_analyzer_par.c`.
2. O arquivo `Makefile`.
3. Um relatório em PDF (`relatorio.pdf`) contendo:
   - Uma breve introdução e descrição da estratégia de paralelização.
   - A tabela com os resultados de desempenho (Tempo, Speedup, Eficiência).
   - O gráfico de Speedup vs. Nº de Threads.
   - Uma **Conclusão** discutindo os resultados. Analise o gráfico de Speedup no contexto de **Escalabilidade Forte**. O speedup obtido foi o esperado? Explique por que o speedup não é perfeitamente linear, relacionando o resultado com a **Lei de Amdahl** e mencionando o conceito de *overhead* de sincronização causado pelo mutex.

Critérios de Avaliação:

- **Corretude (40%):** Os programas compilam e produzem as estatísticas corretas.
- **Implementação Paralela (30%):** Uso correto de Pthreads e do mutex para garantir a segurança da thread.
- **Relatório e Análise (30%):** Qualidade do relatório, apresentação clara dos dados e profundidade da análise na conclusão.

7. Organização, Prazos e Grupos

- **Grupos:** O projeto **deverá** ser realizado em grupos de **3 (três) alunos**. Este tamanho de grupo é ideal para equilibrar a carga de trabalho e garantir que todos os membros participem ativamente da implementação e da análise. Caso algum grupo precise acomodar uma quantidade diferente de alunos, por favor, me procure.
- **Prazo de Entrega:** A data final para a entrega do projeto está definida no Moodle.
- **Apresentação:** Cada grupo deve preparar uma apresentação para demonstrar o que realizou e o que alcançou no desenvolvimento do projeto.
- **Formato de Entrega:** A submissão será de um único arquivo `.zip` contendo todo o material solicitado, via Moodle. Apenas um aluno do grupo precisa realizar a submissão.