Análise da ocorrência de substrings em um texto utilizando POSIX Threads

Rodrigo Karini Leitzke, Thiago Bubolz

Centro de desenvolvimento tecnológico (CDTec) – Universidade Federal de Pelotas (UFPel)
96010-610 – Pelotas – RS – Brazil

{rkleitzke,tlabubolz}@inf.ufpel.edu.br

Abstract. *This paper seeks to compare two algorithms. First a sequential algorithm seeking to find the substring number in a file. And a second algorithm using POSIX Threads, wich considering the number of cores, the tasks are divided in order to reduce the process runtime. At the end of run the runtimes are evaluated speedup.*

Resumo. *Este trabalho tem por objetivo comparar dois algoritmos. Um primeiro algoritmo sequencial que busca encontrar o numero de substrings em um texto, analisando-o linha por linha. E um segundo algoritmo com a utilização de POSIX Threads, onde de acordo com os cores do computador utilizado, o algoritmo divide as tarefas do processo afim de otimizar sua execução. Por fim, os tempos de execução são avaliados com o speedup dos mesmos.*

1. Introdução

Para criação deste trabalho foram implementados dois algoritmos, um primeiro algoritmo sequencial que buscava encontrar a ocorrência de substrings em um texto e um segundo algoritmo utilizando POSIX Threads que dividiu a execução do processo, afim de distribuir tarefas para cara trecho do código. Para ambas as implementações foram utilizadas as bibliotecas *stdio.h, stdlib.h, string.h* e *calcula.h,* sendo a ultima, a biblioteca definida pelas assinaturas das funções utilizadas pelos algoritmos.

2. Desenvolvimento

O presente trabalho pode ser divido em três etapas. A primeira diz respeito a implementação do algoritmo sequencial e a busca por estratégias eficientes para o processo de busca por substrings no texto. Deste modo, após a leitura de uma das linhas do texto, foi realizada uma comparação utilizando a função *const char * strstr (const char * str1, const char * str2)* (ISO/IEC 9899:201x, 2011), da biblioteca string.h, esta função retorna um ponteiro para primeira ocorrência de *str1* em *str2*, ou um ponteiro nulo caso não encontre. Portanto para utilização dessa função, um laço de repetição percorrendo linha por linha do arquivo foi utilizado, considerando que em cada ocorrência o contador de ocorrências fosse incrementado.

A segunda etapa diz respeito a adição das threads e o uso da exclusão mutua no algoritmo sequencial. Para isso foi necessário um processo de adaptação na estratégia utilizada inicialmente. Primeiro, tornou-se necessária a utilização de um buffer (uma

lista circular) contendo um conjunto de linhas do arquivo de texto de entrada. O preenchimento dessa lista se deu com a utilização de uma thread acompanhada de um mutex. Em seguida, outra thread foi acionada para as remoções da lista para a análise de substrings, contendo um mutex. Por ultimo, as demais threads foram utilizadas no processo de contagem das ocorrências, também utilizando o mutex para o controle do acesso ao contador.
Para primeira e segunda etapa, um codigo de testes utilizando a biblioteca *simpletest.h* foi gerado. Nele foram analisadas questões de ocorrência e eventuais situações criticas da execução dos códigos para um determinado arquivo de entrada.
A terceira e ultima etapa foi desenvolvida através da análise dos resultados obtidos e a criação deste relatório afim de documentar o que foi estudado e discutido neste trabalho.

3. Resultados

Para analisar os resultados com s foram considerados os dados médios de 10 execuções para cada algoritmo. Os testes foram realizados em um processador Intel Core i5 3330 3.0GHz com 4 núcleos. Além disso, foi utilizado um arquivo de texto com 8.005.829 linhas, comparando sua execução na busca por uma substring recorrente no corpo do texto (‘ ’), e por uma substring pouco recorrente no corpo do texto (‘material’). Os testes representados nas tabelas a seguir buscam comparar a aplicação com 1, 2 e 3 threads com a versão sequencial da implementação.

Tabela 1. Tabela com os dados médios para cada execução considerando 3 Trheads buscando pela substring: ‘ ’ (espaço).

Tempo	Versão com 3 Threads	Versão Sequencial	Speedup
real	0m17.835s	0m6.234s	2,86
user	0m29.884s	0m6.036s	4,95
sys	0m13.084s	0m0.196s	66,75

Tabela 2. Tabela com os dados médios para cada execução considerando 3 Trheads buscando pela substring: ‘material’.

Tempo	Versão com 3 Threads	Versão Sequencial	Speedup
real	0m19.724s	0m4.918s	4.01
user	0m25.988s	0m4.760s	5.45
sys	0m11.840s	0m0.156s	75.89

De acordo com as tabelas 1 e 2, é possível perceber que o teste aplicado para o código concorrente utilizando 3 threads possui um custo maior em tempo quando comparado a versão sequencial de acordo com o calculo de Speedup (concorrente/sequencial).

Tabela 3. Tabela com os dados médios para cada execução considerando 2 Trheads buscando pela substring: ‘ ’ (espaço).

Tempo	Versão com 2 Threads	Versão Sequencial	Speedup
real	0m15.177s	0m6.234s	2.43
user	0m30.116s	0m6.036s	4.98
sys	0m11.192s	0m0.196s	57.10

Tabela 4. Tabela com os dados médios para cada execução considerando 2 Trheads buscando pela substring: ‘material’.

Tempo	Versão com 2 Threads	Versão Sequencial	Speedup
real	0m19.135s	0m4.918s	3.89
user	0m24.644s	0m4.760s	5.17
sys	0m10.692s	0m0.156s	68.53

Pelas tabelas 3 e 4, podemos perceber que o comportamento do custo em tempo da execução se repete com o uso de 2 threads, quando comparado aos valores obtidos nas tabelas 1 e 2. A versão sequencial apresenta um tempo menor de execução.

Tabela 5. Tabela com os dados médios para cada execução considerando 1 Trheads buscando pela substring: ‘ ’ (espaço).

Tempo	Versão com 1 Thread	Versão Sequencial	Speedup
real	0m16.547s	0m6.234s	2.63
user	0m22.548s	0m6.036s	3.73
sys	0m5.584s	0m0.196s	28.48

Tabela 6. Tabela com os dados médios para cada execução considerando 1 Trheads buscando pela substring: ‘material’.

Tempo	Versão com 1 Thread	Versão Sequencial	Speedup
real	0m17.276s	0m4.918s	3.51
user	0m19.412s	0m4.760s	4.07
sys	0m8.020s	0m0.156s	51.41

Finalmente, para as tabelas 5 e 6, mais uma vez o comportamento se repete. Para o uso de apenas uma thread, a versão sequencial apresenta resultados menores com relação ao tempo de execução, se comparado a versão distribuida.

4. Conclusão

Pelos resultados apresentados pelo presente trabalho é possível concluir que a versão concorrente apresentou um comportamento mais lento que a versão sequencial. Isso se deve a simplicidade da tarefa de encontrar substrings em um texto, já que por condição de corrida e o fato da seção critica estar sempre sendo compartilhada por todas as threads, todas elas passam a esperar por elementos que ingressam na lista, ficando mais tempo paradas do que executando.
Uma solução possivelmente mais rápida para o mesmo problema, seria primeiro realizar a inserção de todas as linhas e após concluído, realizar apenas a tarefa de remoção em paralelo o que evitaria o tempo de espera ocasionado pela inserção.

Referências

Silberschatz, A. (2010) “Fundamentos de sistemas operacionais”. 8a. edicao. Rio de Janeiro: LTD.

Oliveira, R. S., Carissimi, A. S., Toscani, S. S. (2004) “Sistemas Operacionais”. 3. ed. Porto Alegre: Sagra–Luzzatto.

Blaise Barney.(2016) “POSIX Threads Programming”. Lawrence Livermore National Laboratory.
https://computing.llnl.gov/tutorials/pthreads/

ISO/IEC 9899:201x, N1570. (2011) “Programming languages — C”. Committee Draft,
http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1570.pdf