Universidade Federal do Espírito Santo

Departamento de Computação e Eletrônica - CEUNES

Processamento Paralelo

Prof. Oberlan Romão

Exercício Programa 1

Password Leak

O conselho que sempre usamos (pelo menos, deveríamos usar) quando vamos nos cadastrar em algum site é não utilizar a mesma senha para diferentes serviços online. No entanto, nos últimos anos temos nos deparado com várias notícias de vazamento de dados de sites, incluindo o nome do usuário e senha.

Como podemos saber se os nossos dados foram "expostos"? Existem alguns sites especializados em testar se um determinado e-mail está no banco de dados de contas vazadas. Dentre vários, o mais utilizado é o site `https://haveibeenpwned.com/`. Outros sites, por outro lado, verificam se determinada senha é considerada exposta/vulnerável (está no banco de dados de senhas vazadas). Nesse caso, ao digitar uma senha, o site exibe uma mensagem avisando se a senha está ou não na lista de senhas vazadas.

Esse é o objetivo desse trabalho: ler um arquivo com senhas vazadas e outro com senhas que se deseja verificar se foram ou não vazadas. Ao final, seu programa deve imprimir o número de senhas testadas que foram vazadas (localizadas). A quantidade de threads e o nome dos arquivos deve ser passados como argumento para o programa. Por exemplo, considerando que seu programa chama-se `verifica.exe`, a seguinte execução:

```
./verifica.exe 2 senhasVazadas.txt senhasVerificar.txt
```

verifica quais senhas, presentes no arquivo `senhasVerificar.txt`, estão no arquivo `senhasVazadas.txt`, sendo executado com 2 threads. Em ambos os arquivos, a leitura é feita até o fim do arquivo (EOF). Para maior segurança, todas as senhas foram criptografadas usando SHA1. O número de senhas a ser verificada/vazadas será menor ou igual que 10.000.000.

Esse é um problema clássico da computação: busca. A forma mais natural de resolver esse problema é comparar a senha buscada com cada uma das senhas vazadas (*busca sequencial*). Esse algoritmo possui complexidade $O(n)$. Considerando que serão feito m buscas, então a implementação terá complexidade $O(mn)$. Uma forma de melhor a complexidade do algoritmo é ordenar os dados e usar uma *busca binária* ($O(m \log n)$). No entanto, temos o custo computacional para ordenar os dados. Será que paralelizar a busca sequencial, conseguimos um tempo de execução melhor que a busca binária? Para isso, precisamos de quantas threads? Para qual tamanho de n e m? Ou será que ordenar os dados não traz vantagens? Essas e outras perguntas que devem ser respondidas nesse EP.

Tarefa

Você deve implementar dois códigos, um que contém a implementação da busca sequencial e outro com a busca binária, ambos paralelizados e com o mesmo objetivo: contabilizar o número de senhas que foram localizadas em um arquivo contendo várias senhas vazadas.

Para testar e validar o seu código, baixe o arquivo `100SenhasVazadas.txt` (que contém um conjunto de 100 senhas vazadas) e o arquivo `10SenhasVerificar.txt` (que contém 10 senhas a serem verificadas se vazaram ou não). Para esses arquivos, a resposta é 5, ou seja, cinco senhas presentes no arquivo

`10SenhasVerificar.txt` foram expostas (estão presentes no arquivo `100SenhasVazadas.txt`). Teste com diferentes números de threads para ter certeza que tudo está funcionando como esperado. Note que, independente do número de threads, a solução tem que ser sempre 5.

Após ter certeza que seu código está correto, você deve fazer os experimentos considerando:

- O número de threads = $\{1, 2, 3, 4, 8\}$;
- O arquivo com as senhas vazadas: `SenhasVazadas.txt`
- Os arquivos com as senhas a serem verificadas: `1.txt`, `2.txt` `100.txt`, `500.txt`, `1000.txt` e `5000.txt`
- Dados: https://drive.google.com/open?id=1BBADdiMXB2QQMcAik2KSrVJcrBv7W3Q4

O que entregar

- Código (.c ou .cpp) paralelo (usando std::thread ou Pthread) usando o método de busca sequencial;
- Código (.c ou .cpp) paralelo (usando std::thread ou Pthread) usando o método de busca binária;
- Um relatório (.pdf) reportando a estratégia usada para paralelizar os códigos, as configurações do computador utilizados, os experimentos feitos e os gráficos obtidos. Considere o tempo gasto para comparar as diferentes versões. Seu relatório deve conter, no mínimo, uma discussão (gráficos podem e vão ajudar) a respeito das seguintes perguntas (além das mencionadas anteriormente):
  - Considerando a busca sequencial, a partir de que quantidade de senhas é mais vantajoso paralelizar? E considerando a busca binária?
  - Considerando a busca sequencial, a partir de quantos threads o tempo não diminui consideravelmente? E considerando a busca binária?
  - Considerando o número máximo de threads do computador utilizado, teve algum caso em que a implementação paralelo que usa o busca sequencial foi melhor que a que usa a busca binária?

Data de entrega: até às 6h do dia 10/06/2018

Observações:

1. Você deve implementar o código usando a linguagem de programação C/C++ (minha sugestão é usar C++ e std::thread);
2. Para gerar os gráficos e relatório, execute seu programa, no mínimo, 10 vezes;
3. Não envie nada mais que os códigos-fontes e o arquivo do relatório;
4. Recomendo que os experimentos **não** sejam feitos em uma máquina virtual. Preferencialmente, faça os experimentos nos computadores do Laboratório de Computação;
5. Para medir o tempo, use, preferencialmente, a biblioteca chrono[1] do C++ (no arquivo `pratica5.cpp` da Aula Prática 5 tem um exemplo) ou a função `MyClock();`
6. Contabilize o tempo total do seu programa, algo assim (em C++ e usando a biblioteca chrono):

[1] http://www.cplusplus.com/reference/chrono/

```
int main(){
    auto start_time = timeNow();

    ...

    auto end_time = timeNow();
    cout << duration(end_time - start_time) / 1000.f << " ms\n" << flush;
    return 0;
}
```

7. Em caso de plágio, será atribuído 0 a todos os envolvidos.

Critérios de avaliação:

A nota geral do EP será da seguinte forma:

- Implementação dos métodos de busca: 5.0 pontos;
  - Método de busca sequencial paralela: 2.0 pontos;
  - Método de busca binária paralela: 3.0 pontos;
- Relatório técnico: 5.0 pontos (caprichem). O relatório deve conter, pelo menos, os seguintes pontos:
  - Análise e discussão dos resultados obtidos pelos método implementados, mostrando vantagens e desvantagens de cada um;
  - Respostas (em forma de discussão) para as perguntas mencionadas na descrição do EP;
  - Gráficos (preferencialmente, feitos com o `Matplotlib`) comparando os resultados. Sempre que possível, exiba a média e o desvio padrão.

Bom trabalho!!!