MC202GH - ESTRUTURAS DE DADOS

2º Semestre de 2017

Professor: Rafael C. S. Schouery

Monitores: Yulle Glebbyo Felipe Borges (PED)

Caio Vinícius Piologo Véras Fernandes (PAD)

Lab. 09 - Sistema de Autenticação

Peso 4

1. O Problema

Muitos sistemas existentes, principalmente na internet, requerem um processo de autenticação que garanta a identidade de um certo usuário dentro do sistema. Este processo de autenticação deve ser seguro contra usuários maliciosos que procuram por formas de acessar informações confidenciais de outros usuários. Empresas como o Facebook possuem bilhões de usuários que devem ser autenticados no seu sistema todos os dias, por isso, este processo além de seguro, deve ser rápido.

O processo mais simples de autenticação é baseado em um modelo de usuário e senha, no qual cada usuário precisa de um nome de usuário único no sistema e uma senha própria. Para se prevenir contra roubo de informações, geralmente estes sistemas não armazenam as senhas dos usuários em si, mas armazenam o resultado de um hashing criptográfico de cada senha. Isso permite que posteriormente, no processo de autenticação, o sistema aplique a mesma função de hashing criptográfico na senha digitada pelo usuário e compare com a entrada armazenada na base de dados para verificar se o acesso é autêntico.

1.1 Codificação das senhas

Um hashing criptográfico é uma função que mapeia mensagens de uma quantidade arbitrária de caracteres para uma cadeia de bytes de tamanho fixo (hashing). Uma função de hashing criptográfico ideal deve garantir as seguintes propriedades:

- Determinismo: uma mesma mensagem deve retornar sempre o mesmo código de hashing;
- Não reversível: deve ser inviável computacionalmente gerar uma mensagem a partir de seu hashing;
- Resistência a colisões: deve ser improvável duas mensagens gerarem o mesmo hashing.

Existem várias funções de hashing criptográfico que são amplamente utilizadas neste tipo de aplicações, como MD5, SHA-3, etc.

Entretanto, por simplicidade, iremos utilizar neste trabalho a técnica de One Time Pad (OTP) para codificar as senhas. Esta técnica não é considerada uma função de hashing criptográfico boa, pois ela não garante a propriedade de ser não-reversível, já que ela foi criada para aplicações em criptografia onde as mensagens devem ser codificadas e decodificadas com facilidade. Entretanto, ela possui implementação relativamente simples quando comparada à técnicas de hashing criptográfico mais seguras (como MD5 ou SHA-3), por isso escolhemos ela para simular um hashing criptográfico neste exercício.

One Time Pad é uma técnica criptográfica que utiliza uma chave mestra previamente conhecida para codificar mensagens com a mesma quantidade de caracteres que a chave mestra. A codificação é feita caractere a caractere através de uma adição modular.

Assim, considere que temos um alfabeto com 93 símbolos com suas codificações de acordo com a parte imprimível da tabela ASCII menos o caractere espaço branco, uma chave mestra de 5 caracteres, `chave`, e queremos codificar uma senha de 5 caracteres, `senha`. Já que os símbolos considerados estão no intervalo de 33 até 126 (de acordo com seus códigos ASCII) devemos subtrair 33 da codificação de cada caractere, somar os códigos da senha com os da chave mestra, usar o módulo da soma com 94 (126 - 33 + 1) e adicionar 33 ao final para obtermos uma mensagem codificada com caracteres no intervalo de 33 até 126. Para ilustrar, considere o exemplo a seguir, no qual queremos codificar a palavra `senha`:

```
      s     e     n     h     a          palavra
      82    68    77    71    64         palavra ASCII - 33
+
      c     h     a     v     e          chave mestra
      66    71    64    85    68         chave mestra ASCII - 33
_____________________________
     148   139   141   156   132         chave + palavra
mod
      94    94    94    94    94
_____________________________
      54    45    47    62    38         palavra codificada ASCII - 33
+
      33    33    33    33    33
_____________________________
      87    78    80    95    71         palavra codificada ASCII
      W     N     P     _     G          palavra codificada
```

A decodificação pode ser feita através da operação inversa. Além disso, considere que **a chave mestra sempre terá o tamanho máximo de uma palavra válida**, e caso a senha tenha um número de caracteres menor que a chave mestra, **complete a senha com o caractere * (ASCII 42) até que a palavra tenha o mesmo tamanho da chave**. Por exemplo: se tivermos a chave mestra `chavemestra`, e a palavra `senha` para ser codificada, o programa deve codificar a palavra `senha*****`. Ao completar a senha com *, **não se esqueça de adicionar um `\0` ao fim da string**, já que você pode ter sobrescrito o `\0` anterior com um *.

Lembre-se que na linguagem C, o operador % dá o resto de uma divisão inteira, que difere da função modulo algébrico para números negativos. Por exemplo:

```
-2 % 10 = -2
-2 mod 10 = 8
```

1.2 Estrutura da tabela hash

Para armazenar as informações de usuário e senha, deve-se implementar uma tabela hash que utiliza como chave o nome do usuário e armazena sua senha codificada. Com esta estrutura, deve ser possível consultar se existe algum usuário cadastrado com um determinado nome de usuário, e consultar qual a senha codificada de um usuário cadastrado.

Para implementar esta tabela hash, você deve utilizar **o método da divisão** para calcular o hashing. **Para isso, considere o tamanho do hashing M, como 1783**. Adicionalmente, deve-se implementar um **hashing de encadeamento separado**, ou seja, cada posição da tabela tem um ponteiro para uma lista de elementos armazenados naquela posição. Os elementos conflitantes **devem ser adicionados no início** da lista de elementos da posição correspondente ao seu hashing, logo os últimos elementos inseridos aparecem primeiro na tabela.

1.3 Mudança da chave mestra

Uma vez que estamos utilizando uma codificação que depende de uma chave mestra para sua segurança, é natural a necessidade de atualizar esta chave mestra. Entretanto, o codificação de cada senha armazenada na tabela depende da chave mestra, logo se simplesmente alterarmos a chave mestra a autenticação de usuários cadastrados com a chave mestra antiga deixará de funcionar.

Por isso, ao mudar a chave mestra, deve-se utilizar a chave antiga para decodificar todas as senhas cadastradas e re-codifica-las utilizando a nova chave mestra, atualizando os campos de senha codificada para a nova codificação, e passando a utilizar a nova chave mestra para codificação de novas senhas e autenticação.

1.4 Objetivo

O objetivo deste trabalho é criar um sistema de autenticação que aceite a criação de usuários, remoção de usuários, validação de login e mudança de chave mestra. Além disso, deve existir um comando para imprimir todos os usuários e senhas codificadas do sistema. O programa **deve utilizar uma tabela hash com encadeamento separado com método da divisão** para armazenar os dados de login para acesso eficiente, e deve armazenar apenas a senha codificada para cada usuário. **As senhas não codificadas de cada usuário não devem salvas em nenhum tipo de estrutura no seu programa**.

2. Entrada

Ao iniciar o programa, deve-se primeiramente ler a chave mestra inicial do programa. Esta chave é dada na forma de uma string sem espaços brancos. Uma vez lida a chave mestra, o programa entra em um laço de comandos para lidar com os dados de autenticação dos usuários. Estes comandos e suas entradas são descritos nesta seção.

Por simplicidade, assumimos que todas as strings a serem lidas neste trabalho, são vetores de no máximo 40 caracteres e possuem apenas valores da parte imprimível da tabela ASCII de 33 (exclamação, !) até 126 (til, ~), ou seja, não incluem o caractere de espaço branco (ASCII 32). Assumimos também que todas as **chaves mestras devem ter exatamente 40 caracteres**, e que as **senhas dos usuários não usarão o caractere ***.

ID	Nome	Descrição
1	Cadastrar Usuário	Dado uma **string NOME** representando um nome único de usuário, e uma **string SENHA** representando a senha deste usuário, codifica a senha e armazena usuário e senha codificada na tabela de usuários
2	Remover Usuário	Dado uma **string NOME** representando um nome de usuário já cadastrado, remove a entrada correspondente da tabela de usuários
3	Efetuar Login	Dado uma **string NOME** representando o nome de um usuário já cadastrado, e uma **string SENHA** representando a senha deste usuário, codifica **SENHA** e compara com a senha codificada encontrada na tabela de usuários
4	Alterar Chave Mestra	Dado uma **string CHAVE**, altera a codificação de todas as senhas cadastradas para serem codificadas com **CHAVE**, efetivamente fazendo que **CHAVE** seja a nova chave mestra
5	Relatório	Para cada usuário cadastrado, imprime seu nome de usuário e senha codificada. Ver Seção 3 para mais informações sobre a formatação
0	Finalização	Saia do sistema, certificando-se que toda a memória alocada foi liberada.

Exemplo de entrada:

```
chave
1 usuario senha
3 usuario sinha
```

```
3 usuario senha
5
2 usuario
3 usuario senha
0
```

Obs: O resultado do exemplo acima pode diferir de sua implementação, uma vez que ele foi construído a mão para facilitar seu entendimento, logo consideramos chaves de tamanho 5. Para exemplos mais detalhados ver os casos de teste abertos no SuSy.

3. Saída

A tabela abaixo descreve o formato esperado da saída de cada comando definido na seção anterior.

ID	Saída	Comentário
1	Usuario cadastrado com sucesso!	Caso não exista uma entrada na tabela de usuários com o mesmo nome de usuário da entrada, um usuário com este nome deve ser cadastrado
	Usuario ja existe!	Caso já exista uma entrada na tabela de usuários com o mesmo nome de usuário da entrada, nada deve ser feito
2	Usuario removido com sucesso!	Caso exista uma entrada na tabela de usuários com o nome dado como entrada, remove este usuário
	Usuario inexistente!	Caso não exista uma entrada na tabela de usuários com o nome dado como entrada, nada deve ser feito
3	Autenticacao realizada com sucesso! Bem vindo, USUARIO.	Caso exista uma entrada na tabela de usuários com o nome dado como entrada, e sua senha codificada é igual a senha de entrada codificada, nada deve ser feito. USUARIO deve ser substituído pelo nome do usuário autenticado.

	Nome de usuario ou senha incorretos!	Caso não exista uma entrada na tabela de usuários com o nome dado como entrada, ou a senha de entrada codificada não seja igual a senha codificada armazenada na tabela, nada deve ser feito
4	Chave mestra alterada com sucesso!	Altera a chave mestra de codificação, consequentemente alterando a codificação de cada senha cadastrada.
5	Usuario: USUARIO1\tSenha: SENHA1 Usuario: USUARIO2\tSenha: SENHA2 … Usuario: USUARION\tSenha: SENHAN	USUARIO1, USUARIO2, USUARION são os nomes de usuários cadastrados, e SENHA1, SENHA2, SENHAN são suas senhas codificadas respectivamente. Os campos de usuário e senha são separados por um caractere de tabulação (`\t`). Temos um usuário e senha por linha, e após o último usuário temos um `\n`
0	Sistema encerrado.	Antes de encerrar o sistema, é necessário liberar toda a memória alocada

Exemplo de saída:

```
Usuario cadastrado com sucesso!
Nome de usuario ou senha incorretos!
Autenticacao realizada com sucesso! Bem vindo, usuario.
Usuario: usuario    Senha: WNP_G
Usuario removido com sucesso!
Nome de usuario ou senha incorretos!
Sistema encerrado.
```

Obs: O resultado do exemplo acima pode diferir de sua implementação, uma vez que ele foi construído a mão para facilitar seu entendimento, logo consideramos chaves de tamanho 5. Para exemplos mais detalhados ver os casos de teste abertos no SuSy.

4. Informações

- Este laboratório possui **peso 4**.
- **Não** há um número máximo de submissões.

- No início de cada arquivo a ser submetido, insira um comentário com seu nome, RA e uma breve descrição do conteúdo do arquivo.
- Apenas comentários no formato `/* comentário */` serão aceitos. Comentários com `//` serão acusados como erros pelo SuSy.
- Os cabeçalhos permitidos são:
    - `stdio.h`
    - `stdlib.h`
    - `string.h`
    - `math.h`
- A submissão da sua solução deverá conter múltiplos arquivos:
    - **tabela_hash.h**: interface da estrutura de tabela hash
    - **tabela_hash.c**: implementação da estrutura de tabela hash
    - **lab09.c**: cliente que utiliza a estrutura
- Será disponibilizado um **Makefile** para este trabalho na página do laboratório:
    - Para compilar seu projeto, basta navegar até o diretório do projeto e utilizar o comando ‘make’ no terminal do Linux.
    - Veja mais instruções na página da disciplina.
- Toda a memória alocada deve ser liberada adequadamente ao fim do programa. Isso pode te ajudar a evitar a ocorrência de falhas relacionadas a memória.
- Indente corretamente todo o seu código. Escolha entre espaços ou tabs para indentação e seja coerente em todos os arquivos. Indentação é fundamental para a legibilidade do seu código, e ajuda no entendimento do fluxo de controle do seu programa.

5. Critérios de Avaliação

- Seu código deverá passar por todos os casos de teste do SuSy definidos para este laboratório. Caso positivo, utilizaremos os seguintes critérios adicionais:
    - A **não utilização/implementação da estrutura de tabela hash com encadeamento separado e método da divisão** resultará em **nota zero**

- Falha na implementação do comportamento esperado para a tabela de hash acarretará em uma penalização
- Utilização de bibliotecas ou funções não permitidas pelo enunciado resultará em uma **penalização severa** na nota