UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO

INTRODUÇÃO À CIÊNCIA DA COMPUTAÇÃO - TRABALHO 04: "REDE SOCIAL"

Gustavo Romanini Gois Barco
10749202
Isabella Ferreira Lopes
10872281
Lucas Alves Roris
11913771

SÃO CARLOS
Junho de 2020

1. INTRODUÇÃO À ATIVIDADE

Nesta tarefa, foi proposto o desenvolvimento de uma rede social com funcionalidades similares às do “Twitter”. Os principais tópicos da linguagem C utilizados foram arquivos, ponteiros, alocação dinâmica, Makefile e estruturas de dados. Os requisitos deste exercício foram:

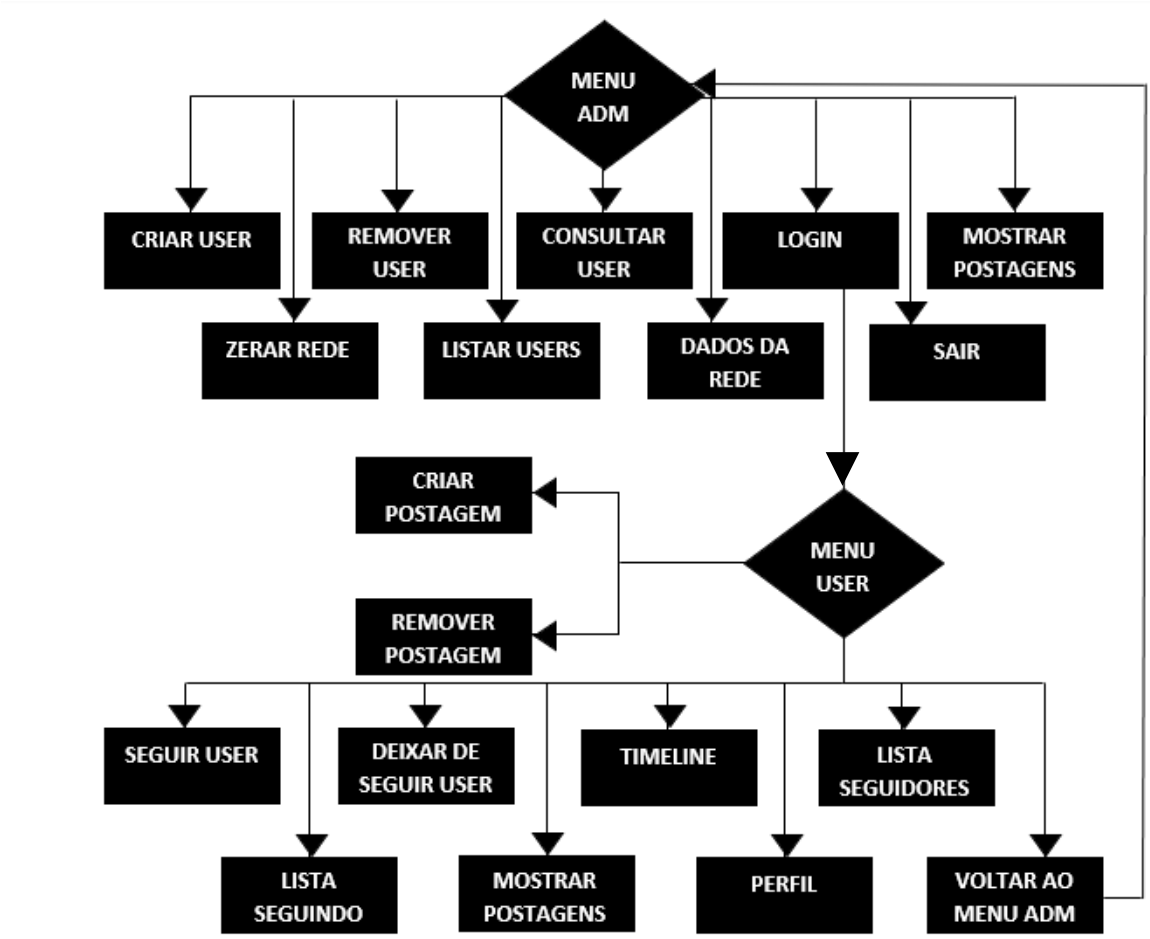
- Menu de Administrador com possibilidade de consultar e listar informações gerais sobre a rede e postagens, inserir e remover usuários, excluir todos os dados da rede, fazer login como usuário e sair do programa;
- Menu de Usuário com possibilidade de seguir e deixar de seguir outros, fazer postagens e removê-las, consultar seu perfil e timeline e retornar ao menu de administrador;
- Armazenamento das informações entre as execuções do programa.

2. LÓGICA

A primeira importante decisão tomada pelo grupo foi relativa à manipulação dos dados: foi decidido que, ao invés de tratar as informações diretamente dos arquivos, seria melhor criar structs, que, ao início da execução do programa, receberiam as informações dos arquivos. Durante o uso da rede social, todas as alterações seriam feitas diretamente nas structs, e salvas nos arquivos através da função **Salvar**.

Foram criados seis arquivos do tipo ".C" para separar o código em blocos de funcionalidade similar, para facilitar a visualização, o entendimento e, por fim, a concretização do Makefile.

O fluxo planejado é apresentado a seguir.



3. DESENVOLVIMENTO

A disposição de dados do programa foi feita da seguinte forma:

- O arquivo **struct.h** contém as duas structs em *TYPEDEF* primordiais para o programa. A primeira é a "dados", que armazena informações dos usuários, e a segunda é a "dadosPosts", que armazena informações das postagens. Os elementos são:
    1. Dados: nickname, nome, número de postagens, número de pessoas que segue, vetor de pessoas que segue, número de seguidores, vetor de seguidores;
    2. DadosPosts: nickname do autor, postagem.

Esse arquivo foi definido no cabeçalho de cada parte do código do programa, pois praticamente todas as funções precisam dos dados dessas estruturas. Por estarem definidas em *TYPEDEF*, foram criados os ponteiros *usuarios (do tipo dados) e *posts (do tipo dadosposts), esses ponteiros foram utilizados para as manipulações de informação. Para o armazenamento de informações, foram criados os arquivos Usuario.txt e Posts.txt

As funções do programa estão dispostas da seguinte forma dos arquivos.C:

- **ALOCACAO.C**

  Armazena as funções que verificam, a cada alocação ou realocação de ponteiros, se o retorno foi igual a NULL. Nesse caso, o programa é abordado e uma mensagem é apresentada ao usuário. São cinco funções, e cada uma analisa ponteiros de um dos tipos: ponteiro de int, ponteiro de char, ponteiro de ponteiro de char, ponteiro de dados e ponteiro de dadosPost.

- **FUNCOESBD.C**

Aqui foram colocadas as funções que tratam, de maneira direta, os dados e as traduções entre struct e arquivos. O nome se refere a "funções de banco de dados", e elas são:

1. **Inicializar**: traduz (pela função fread) e transfere (por laços de repetição com os elementos das structs) os dados dos arquivos para as structs de usuários e dados;

2. **Salvar**: utiliza a função fwrite para escrever nos arquivos todos os elementos das structs, seguindo um padrão para que, futuramente, possam ser acessados na devida ordem;
3. **Terminar**: usa a função free para liberar os ponteiros do programa;
4. **ZerarRedeSocial**: pede confirmação do usuário, e se a resposta for positiva, remove os arquivos de postagens e usuários e cria novos, vazios, limpando, assim, os dados da rede social;
5. **VerificarExistencia**: recebe um nickname como parâmetro e percorre a struct de usuários para checar a validade daquele dado. Caso exista, retorna o número correspondente àquele usuário, do contrário, retorna -1;
6. **MostrarInformacoes**: printa os números totais de usuários e postagens da rede social e apresenta o(s) usuário(s) com maior número de seguidores.

- **INTERFACE.C**

Neste arquivo ficam as funções MenuSLogin e MenuCLogin, que printam as interfaces dos menus anterior e posterior ao login, respectivamente. Essas funções são chamadas dentro de cada um dos dois laços principais da Main.

- **POST.C**

Nesse arquivo foram desenvolvidas todas as funções que manipulam, diretamente, as postagens na rede social, seja essa manipulação feita pelo menu de administrador ou pelo de usuário. São elas:

1. **AdicionarPost**: antes de ser chamada, é feita a realocação e a verificação do ponteiro *Posts. Essa função printa uma mensagem ao usuário, lê a mensagem que ele quer publicar, verifica se ela não ultrapassa o limite de caracteres (127), trata o erro caso isso aconteça, e no caso contrário, transfere a mensagem inserida à última posição do ponteiro, incrementando o valor total do número de postagens;
2. **RemoverPost**: primeiramente, essa função printa ao usuário, em ordem, todas as postagens que ele já fez, sendo que cada uma delas corresponde a um número, é pedido que o usuário digite o número correspondente à postagem que ele quer remover. Isso acontece se a variável parâmetro “chave” for igual a zero, pois isso

significa que é o usuário que está excluindo seu post. Se a chave for igual a um, a função está sendo chamada por outra função, o que implica que essa primeira parte, de printar as postagens, pode ser pulada. Essa função passa todas as postagens posteriores à escolhida para uma posição anterior, suprimindo a selecionada. Por fim, o número de postagens é decrementado. Há tratamento, como em todas as funções, para entradas inválidas;

3. **MostrarPostsUser**: esta é a função responsável por apresentar aos usuários sua timeline. Ela percorre todas as postagens da struct Posts, verificando o nickname do autor da publicação. Se esse nick for compatível com alguém que o usuário logado está seguindo, a publicação é printada. Do contrário, o laço segue sem fazer nada. Há um contador para cada mensagem publicada. Quando esse contador atinge cinco, é perguntado ao usuário se ele quer ver mais publicações;
4. **MostrarTodosPosts**: esta é a função responsável por apresentar ao administrador todos as publicações da rede. Ela percorre toda a struct de postagens printando as mensagens e os autores, e tal como na **MostrarPostsUser**, apresenta de cinco em cinco;
5. **MostrarPostsAutoria**: é verificado o elemento NPosts (número de postagens) da posição do ponteiro de struct correspondente ao usuário que está logado. Se esse número for zero, o usuário é informado de que não há publicações para mostrar. Do contrário, é printado todo o ponteiro de postagens desse usuário (usuário[x].Posts).

- **USUARIO.C**

Este é o maior arquivo do programa, e aqui ficam as funções referentes aos dados de usuário.

1. **InserirUsuario**: antes de essa função ser executada, o ponteiro de struct de usuários é realocado para mais um espaço. É feita a verificação desse ponteiro, e a primeira ação que esta função executa é pedir ao administrador que insira o nickname do usuário que ele quer criar. A partir daí, há duas verificações. Primeiramente, o nickname deve ter um número máximo de 40 caracteres. Se o administrador tenta inserir mais que isso, o programa não permite. Se o nome passa por essa fase de verificação, o programa percorre todo o ponteiro de struct de usuários verificando se algum outro tem o mesmo nickname que foi inserido,

pois isso também não é permitido. Passada essa fase, é pedido o nome desse usuário, também com limite de 40 caracteres, e os elementos NPosts, NSeguidores (número de seguidores) e NSeguindo (número de pessoas que está seguindo) desse novo usuário são inicializados em zero. O número de usuários é incrementado em 1;

2. **RemoverUsuario**: a função pede que o administrador insira o nickname do usuário a ser removido e chama a função VerificarExistencia para encontrar esse usuário, caso ele exista. É chamada a função RemoverPost, num laço com limite do número de postagens desse usuário, com o parâmetro "chave" equivalendo 1, para que as mensagens de remoção de publicação não sejam printadas, e, para finalizar a exclusão do usuário, é chamada a função ExplodindoUsuario, que será descrita a seguir;
3. **ExplodindoUsuario**: tem a funcionalidade de remover os efeitos de um usuário que foi excluído nos dados dos demais usuários. Por exemplo, quando um usuário é removido, é preciso descobrir todos os usuários que ele seguia, e removê-lo da lista de seguidores deles. Em dois laços principais que percorrem todos os usuários [i] da rede social, são impostas duas condições: se o usuário [i] era um dos seguidores do usuário excluído, há um laço que transfere todos usuários posteriores que [i] está seguindo para uma posição anterior, e isso suprime o usuário excluído dessa lista. Se o usuário [i] era seguido pelo que foi excluído, o mesmo acontece com a lista de seguidores de [i];
4. **Seguir**: essa função pede que o usuário insira o nickname da pessoa que quer seguir. É verificado se essa pessoa existe, se ela não é o próprio usuário logado e se o usuário já segue essa pessoa, passadas as primeiras verificações, são conferidos o vetor de seguidores do usuário que será seguido, e o vetor "seguindo" do usuário que quer seguir. Esses são realocados de cinco em cinco, e, se for o momento propício para realocação, isso é feito. Por fim, esses vetores recebem os respectivos nicknames;
5. **DeixarSeguir**: tal como as anteriores, esta função trata os casos de entradas invalidas, e exerce procedimento oposto à função Seguir, removendo, por laços de repetição, o nickname do usuário que está deixando de seguir do vetor de seguidores do usuário selecionado, e a ação inversa;
6. **ListarSeguidores**: essa função utiliza, predominantemente, o ponteiro "usuário[n].seguidores", printando todos os elementos por um laço For;

7. **ListarSeguindo**: exerce o mesmo procedimento da função ListarSeguidores, usando o ponteiro “usuário[n].seguindo”;
8. **ConsultarUsuario**: recebe do administrador um nickname e percorrendo o vetor “usuário”. Quando encontra um nickname compatível, printa as informações da struct desse usuário: nome, número de postagens, número de seguidores, lista de seguidores, número de pessoas que segue e lista das pessoas que segue;
9. **ListarUsersEInfos**: executa o mesmo procedimento da função ConsultarUsuario, mas não recebe nenhuma entrada e apresenta as informações de todos os usuários;
10. **MostrarInfoUser**: esta função corresponde ao perfil do usuário, e printa as mesmas informações que a função ConsultarUsuario, sendo somente para o usuário logado.

Na função principal, os arquivos **Usuario** e **Posts** são abertos pela função **fopen**. Em Seguida, a função **ftell** é utilizada para buscar dois dos parâmetros primordiais da inicialização do programa: o número de usuários existentes e o número de postagem. Esses números são armazenados em ponteiros inteiros. O motivo da escolha delas como ponteiros é que serão manipuladas pela maioria das funções do programa.

Os ponteiros **usuarios** e **posts**, que serão dinamizados, são inicializados como NULL, e depois realocados de acordo com os números anteriormente buscados. A cada realocação, e chamada a função **VerificarAlocacao** correspondente. Nesse ponto, é chamada a função **Inicializar**.

São criadas as variáveis **x** e **y**, que serão lidas do usuário enquanto ele não fechar o programa. **x** lê a opção referente ao menu de administrador, e y referente ao de usuário. Há tratamento para opções inválidas, e cada opção corresponde a um número. Quando esse número é inserido, é chamada a função correspondente. No menu de administrador, a opção 9 corresponde a sair do programa, e nenhuma função é chamada, o que acontece é a quebra do laço While principal.

4. FINALIZAÇÕES

Para aprimorar a navegação pelo programa, a cada função executada foi dado um comando para limpar a visualização da tela. Esse comando é o System(“clear||cls”), que funciona no sistema operacional Linux e no Windows.

O grupo encontrou situações em que, quando não era mais desejado inserir um nome, por exemplo, não havia maneira de retornar, o programa “obrigava” o usuário a terminar a inserção do dado para que pudesse escolher outra opção. Para isso, foram implementados laços nas funções para que o padrão ” -1 ” pudesse ser digitado para retornar.

Depois de muito tempo sem a definição de um nome, foi escolhido denominar a rede social de “BlaBlaBla”.

5. VÍDEOS

O grupo gravou uma vídeo-chamada de explicação de algumas resoluções do programa, e um segundo vídeo de apresentação da rede social. Abaixo, os links para esses vídeos.

SOLUÇÃO

APRESENTAÇÃO