Trabalho Prático 1
Algoritmos e Estrutura de Dados 2

Geraldo Arthur Detomi, Rhayan de Sousa Barcelos, Rodrigo José Isidoro Junior

1 Introdução

Esse trabalho prático consiste na implementação de um servidor de e-mails, no qual é possível gerenciar usuários e mensagens entre eles. O objetivo consiste em aplicar técnicas aprendidas para que o problema proposto seja solucionado de forma eficiente.

O programa funciona corretamente, alocando e desalocando memória quando necessário, além de conseguir realizar todas as operações exigidas. Isso é feito através do uso de listas encadeadas com *arrays* dinâmicos, tipos abstratos de dados e do uso de encapsulamento para um código mais legível.

2 Implementação

2.1 Análise Geral

Para esse trabalho foram definidos 2 tipos abstratos de dados, sendo esses, o tipo lista encadeada, que é utilizado para armazenar os usuários do sistema, o tipo usuário, que armazena os IDs e uma referência para sua respectiva caixa de entrada, que, por sua vez, é representada por uma lista dinâmica com *array*.

Para armazenar os usuários foi utilizado lista encadeada, pois como as operações seriam apenas de cadastro e remoção, a complexidade para cadastrar, por exemplo, seria O(1). Como exigia pouca manipulação, não fazia diferença ter que lidar com ponteiros.

Agora, para caixa de entrada foi definido uma *struct e-mail*, para armazenar um inteiro para representar a prioridade do *e-mail* e um ponteiro para a mensagem que é alocado dinamicamente conforme o tamanho da mensagem passada, foi escolhido o tipo de lista com *array* dinâmico, pois desse modo pode se ter diversos e-mails sem um tamanho predeterminado, além de que por o *array* ser contíguo e ser manipulável por índice, colocar uma mensagem na posição conforme a prioridade fica muito mais fácil. Além de que um usuário pode ter muitos *e-mails*, mas cada *e-mail* corresponde apenas a um usuário, logo espera-se armazenar mais *e-mails* do que usuários e usar memória contígua para dados maiores oferece maior desempenho.

Também foram definidos cabeçalhos para execução de operações do servidor que envolve a leitura dos arquivos, o que cada código de resposta representa e cada operação que o servidor faz, utilizando *enums* para melhor legibilidade do código. Foram utilizados cabeçalhos que possuem funções gerais que são compartilhadas por todos os outros módulos, para melhor compactação do código e evitar escrita desnecessária.

2.2 Funções

Foram definidos 6 módulos(bibliotecas) separados em arquivos de cabeçalhos e operações, relativos às definições e operações de usuários, de e-mail, de listas encadeadas e listas por *array* e um contendo definições de funções utilitárias."

O programa conta com diversas funções, que foram organizadas em um *.header* (.h) e um .c, sendo sua assinatura, ou seja, o que ela faz colocada no .h e sua implementação, no .c. Essa organização é essencial, pois como existem muitas funções no programa, seria confuso e tornaria a programação mais difícil e cansativa.

2.3 Funções básicas de usuários

Cabeçalho : usuários_operações.h

- **Cadastrar_novo_Usuário:**
  Essa função recebe como parâmetros a referência para a lista de usuários e um inteiro correspondente ao ID do usuário que será inserido na lista e adicionado o novo usuário a lista, caso o ID ainda não esteja cadastrado. A função ao receber os parâmetros faz diversas checagens, primeiro se o ID passado é válido, depois verifica se é corresponde a um usuário já cadastrado. A função tem complexidade O(n), porque mesmo que tenha complexidade O(1) para adicionar um elemento na lista encadeada, por conta de adicionar no inicio, ao fazer a checagem se já existe o ID, é necessário percorrer toda a lista verificando elemento por elemento. E caso se passe uma lista vazia, ela possui complexidade O(1), pois não é necessário verificar se o elemento está presente. E, ao cadastrar um usuário, é setado seu ID em questão. Além de inicializar sua caixa de entrada que utiliza uma lista por *array*.

- **Remover_usuário :**
  Essa função recebe como parâmetros a referência para a lista de usuários e um inteiro que é o ID do usuário. Então verifica se é válido, se a lista está vazia e se a conta existe. Passando por essas verificações sem dar erro, a conta será removida. Essa função primeiro verifica se o ID passado é válido, ou seja, se é maior que 0 e menor que 10000, também verifica se a lista está vazia e retorna erro caso esteja. A função verifica se o elemento está presente e caso esteja retorna que a conta não existe. Também é necessário carregar o usuário que será removido, que no caso, é percorrer a lista e passar o usuário por referência para que possa liberar a memória de

sua caixa de entrada, utilizando a função destroi_lista_array(), que libera a memória de toda caixa de entrada e a mensagem alocada para toda mensagem de um email.

2.4 Funções básicas de email

- **Enviar_email_para_o_usuário:**
  Essa função recebe como parâmetros uma lista de usuários, o ID que identifica o usuário, a mensagem a ser enviada e a prioridade da mensagem. Então verifica se o ID da mensagem é válido, se a prioridade da mensagem é válida e se a lista está vazia. Caso esteja, retorna um erro. Em seguida, verifica se a conta do usuário existe e entrega a mensagem.

- **Consulta_id_msg_priori:**
  Essa função recebe como parâmetros a lista de usuários, o ID que identifica o usuário e o *e-mail* a ser consultado. Então verifica se o ID é valido e se a lista está vazia. Caso esteja, retorna um erro. Em seguida, verifica se a conta do usuário existe e se a caixa de entrada está vazia. Caso esteja, não será possível ler a mensagem. Caso esteja tudo certo a mensagem é consultada.

- **Remove_email_já_consultado:**
  Essa função recebe como parâmetros apenas a lista de usuários e o ID do usuário. Então, se o ID do usuário não for válido ou se a lista estiver vazia, a função se encerra (*return*). Caso contrário, o *e-mail* já lido do usuário selecionado será excluído.

2.5 Memória

O servidor de *e-mails* necessitou de muita alocação e desalocação de memória, já que foi preciso implementar as opções de cadastrar usuário - alocar memória - e remoção do usuário - desalocar memória. Sendo assim, para implementar isso de forma eficaz, foi necessário usar alocação dinâmica, já que uma alocação estática não daria certo. Além disso, para executar certas funções como consultar as mensagens seguindo a ordem de prioridade e então removê-la para ler a próxima, foi necessário o uso de um auxiliar para guardar o conteúdo da próxima mensagem e então desalocar (excluir) a mensagem já lida.

3 Resultados

O desenvolvimento do trabalho ocorreu como o esperado e os artefatos resultantes são funcionais. Foram realizados testes utilizando implementações de lista encadeada e lista por *array*, assim como testes das funcionalidades criadas para as demais estruturas. A ferramente valgrind também foi utilizada para avaliar o manejamento de memória, que não mostrou vazamentos na execução.

Também foram realizados testes utilizando "*sanitize address*", mas estava dando erro devido à utilização da função realloc(). O erro ocorreu, pois ao utilizar *realloc*(), o ponteiro pode trocar de posição. Os testes realizados estão presentes na pasta "testes"do trabalho.

4 Conclusão

A abstração dos dados é essencial para programar de forma eficiente, principalmente quando se trata de desenvolvimento em equipe. O conceito de encapsulamento facilita que diferentes funcionalidades sejam desenvolvidas simultaneamente sem que comprometa o funcionamento do projeto, além de proporcionar uma leitura facilitada. Junto a esses conceitos, o estudo de diferentes estruturas e da complexidade de tempo de execução atrelado às suas funções contribuiu para o conhecimento da equipe e, consequentemente, para a elaboração de um trabalho funcional.

4.1 Dificuldades

Não houve grandes problemas para elaboração e execução do trabalho prático. Apenas um pouco de insistência para a resolução de pequenos *bugs* que ocorreram ao longo do trabalho. Principalmente pois ao definir a mensagem como um tamanho dinâmico ocorreram diversos bugs para desalocar memória no qual ao desalocar uma mensagem estava corrompendo outras, mas ao olhar a documentação e encontrar funções da linguagem mais eficazes, deu para realizar as devidas correções.

5 Bibliografia

CPLUSPLUS, 2023. Disponível em: https://cplusplus.com/reference/ Acesso: em 30 abr. 2023.